Наука и Образование
МГТУ им. Н.Э. Баумана
Сетевое научное издание
ISSN 1994-0408

Наука и Образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2015. № 04. С. 189–214.

DOI: **10.7463/0415.0764268**

Представлена в редакцию: 25.02.2015
Исправлена: 04.04.2015



**УДК 004.2; 004.31**

# Об организации параллельной работы некоторых алгоритмов поиска кратчайшего пути на графе в вычислительной системе с многими потоками команд и одним потоком данных

**Подольский В. Э.**[1,*]

[*]v.e.podolskiy@gmail.com

[1]МГТУ им. Н.Э. Баумана, Москва, Россия

В работе рассматривается реализация алгоритмов Беллмана-Форда и Ли поиска кратчайшего пути в графе для вычислительной системы ЭВМ с многими потоками команд и одним потоком данных. В статье показано прикладное значение алгоритмов Беллмана-Форда и Ли на примере решения задачи планирования движения автономных робототехнических комплексов. Впервые получена версия алгоритмов поиска кратчайшего пути в режиме сопроцессора МКОД, а также параллельная версия (поток центрального процессора и поток процессора структур) в режиме МКОД. Представлены результаты анализа работы алгоритмов в режиме МКОД, сформулированы направления развития методики распараллеливания алгоритмов на поток обработки данных и поток обработки структур, в частности, обозначены перспективы использования математического подхода в описании алгоритмов для автоматизации процесса распараллеливания алгоритмов на потоки.

**Ключевые слова**: алгоритм Беллмана-Форда, алгоритм Ли, много потоков команд и один поток данных, МКОД, процессор обработки структур

## Введение

Класс алгоритмов поиска кратчайшего пути от одной вершины графа до другой вершины находит широкое применение при решении различного рода транспортных задач. В частности, серьёзная роль отводится алгоритмам поиска кратчайшего пути на графе при планировании и организации перемещения робототехнических комплексов (РТК) [1-3].

Несмотря на то, что управление РТК может осуществляться оператором РТК удалённо (например, за счёт средств дистанционного управления), подобная практика имеет ряд недостатков, включающих ненадёжность каналов связи в условиях пересечённой местности, к примеру, городской, а также слабую защищённость канала

управления роботом. В связи с этим, в рамках задачи управления РТК возникает проблема организации полностью автономного передвижения РТК. Эту проблему позволяет решить совместное функционирование системы технического зрения (СТЗ) РТК, системы формирования модели внешней среды (СФМВС) и системы планирования движения (СПД) РТК, работа которой основана на обработке графовых структур. Более подробная схема автономной системы управления движением РТК приведена на рис. 1.

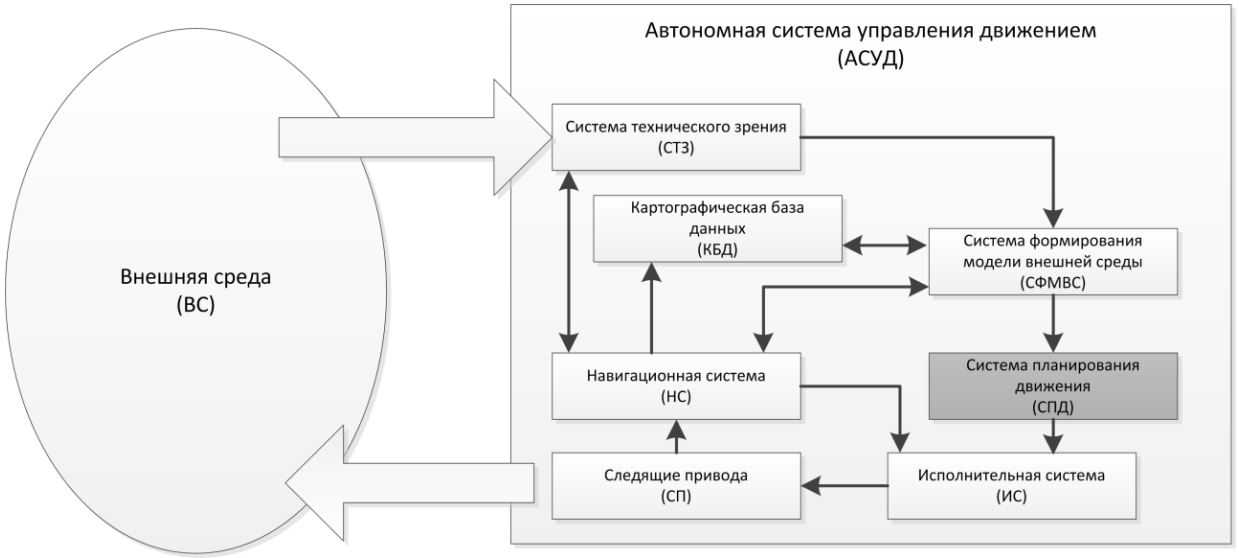



**Рис. 1.** Структурная схема автономной системы управления движением

В процессе планирования движения РТК одной из ключевых решаемых системой планирования движения задач является задача поиска кратчайшего расстояния в глобальном графе, сформированном СФМВС РТК. Само решение задачи формирования глобального графа с учётом разнообразных препятствий и динамических изменений внешней среды представляет собой самостоятельную ценность для реализации функции автономного передвижения РТК, но выходит за рамки настоящей работы. На рис. 2 приведён пример типичного графа, формируемого СФМВС РТК на основе данных, полученных от системы технического зрения РТК.

Реализация функции планирования движения РТК основывается на поиске кратчайшего пути между вершинами в глобальном графе, сформированном СФМВС РТК. Поскольку глобальный граф может представлять собой структуру весьма большой размерности, а задача планирования движения РТК в условиях меняющейся внешней среды носит характер задачи реального времени, весьма актуальной оказывается возможность увеличения скорости поиска кратчайшего пути в графе. Для решения этой задачи в робототехнике часто применяются алгоритмы Беллмана-Форда и Ли.

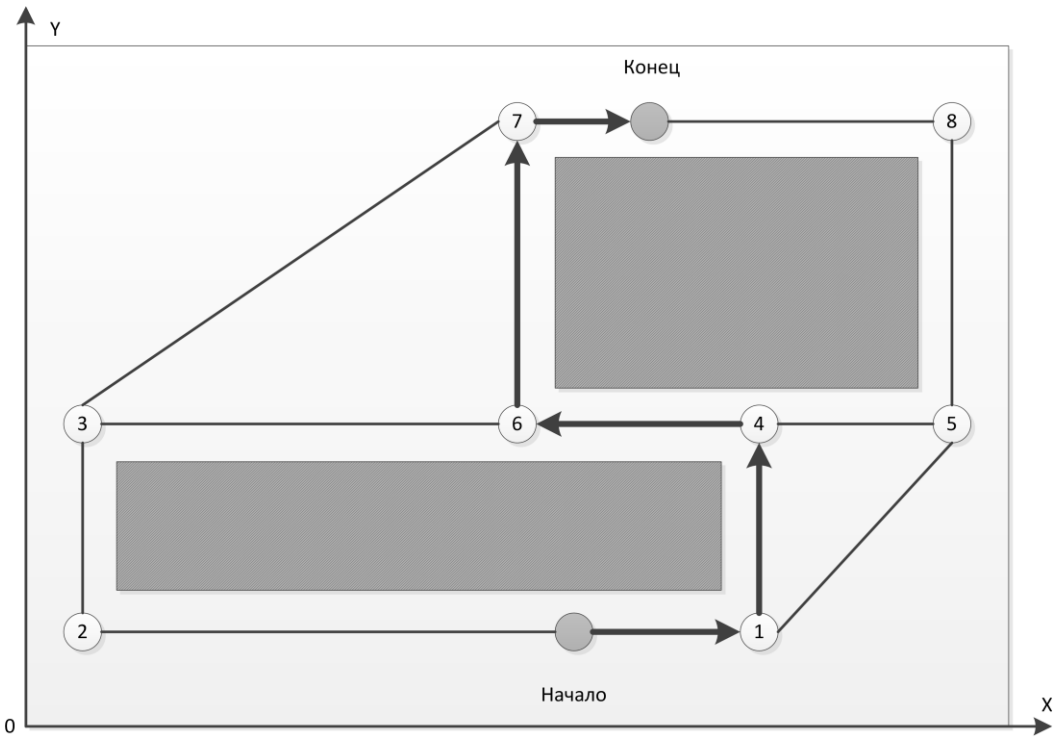



**Рис. 2.** Глобальный граф на плане (затемненные прямоугольники – препятствия)

**Обзор ранних работ** показал, что было разработано большое количество последовательных вариантов алгоритмов поиска кратчайшего пути в графе [4-6]. Последовательные версии данных алгоритмов не использовали весь потенциал параллелизма, заложенный в алгоритмах обработкой структур данных, а потому показывали недостаточную для оперативного решения многих задач реального времени производительность [7]. Тем не менее, отдельные алгоритмы на графах были модифицированы для параллельной архитектуры ЭВМ МКОД [8,9]. В частности, алгоритм Дейкстры показал весьма хорошую способность к распараллеливанию на несколько потоков команд при условии обработки одного потока данных на двух процессорах [8].

**Настоящая работа** призвана продолжить научные изыскания в сфере поиска и модификации для архитектуры ЭВМ МКОД последовательных алгоритмов, в которых происходит интенсивная обработка структур данных. Помимо этого, статья одной из первых увязывает модификацию конкретных алгоритмов для ЭВМ МКОД с решением прикладной задачи (планирование движения автономных робототехнических комплексов).

**Цель настоящей работы** заключается в модификации алгоритмов поиска кратчайшего пути в графе (алгоритмы Беллмана-Форда и Ли) для архитектуры ЭВМ со многими потоками команд и одним потоком данных (ЭВМ МКОД) с процессором обработки структур (далее – СП), а также в выявлении путей оптимизации процесса модификации алгоритмов для ЭВМ МКОД. Эффективность ЭВМ МКОД с процессором

обработки структур при решении подобных задач ранее была продемонстрирована на примере алгоритмов Дейкстры [8] и Форда-Фалкерсона [9].

**Научная новизна настоящей работы** состоит в том, что алгоритмы поиска кратчайшего пути в графе от одной вершины до другой (в том числе алгоритмы Беллмана-Форда и Ли) никогда прежде не были адаптированы для ЭВМ с многими потоками команд и одним потоком данных. Существующие версии указанных алгоритмов широко применяются в робототехнике лишь в своём последовательном варианте.

**Практическая ценность** адаптации алгоритмов поиска кратчайшего пути в графе состоит в получаемом аппаратном ускорении при решении задачи планирования движения робототехнического комплекса.

Адаптация алгоритмов поиска кратчайшего пути Беллмана-Форда и Ли для ЭВМ МКОД осуществлена для разработанной в МГТУ имени Н.Э. Баумана вычислительной системы со многими потоками команд и одним потоком данных.

## 1. Алгоритмы поиска кратчайшего пути в графе

Задача поиска кратчайшего пути — задача поиска самого короткого пути (цепи) между двумя точками (вершинами) на графе, в которой минимизируется сумма весов ребер, составляющих путь [10,11]. Для решения задачи поиска кратчайшего пути в графе разработан ряд алгоритмов, отличающихся вычислительной сложностью и отдельными особенностями функционирования и представления результатов:

- алгоритм Дейкстры (находит кратчайший путь от одной из вершин графа до всех остальных, но выдаёт корректный результат только для графов без рёбер отрицательного веса);
- алгоритм Беллмана — Форда (находит кратчайшие пути от одной вершины графа до всех остальных во взвешенном графе, при этом вес ребер может быть отрицательным);
- алгоритм поиска A* (находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной), используя алгоритм поиска по первому наилучшему совпадению на графе);
- алгоритм Флойда — Уоршелла (находит кратчайшие пути между всеми вершинами взвешенного ориентированного графа);
- алгоритм Джонсона (находит кратчайшие пути между всеми парами вершин взвешенного ориентированного графа);
- алгоритм Ли (основан на методе поиска в ширину. Находит путь между вершинами s и t графа (s не совпадает с t), содержащий минимальное количество промежуточных вершин (ребер));
- алгоритм Килдала.

Обычно выбор необходимого алгоритма базируется на том, как поставлена задача поиска кратчайшего пути. Постановки задачи о кратчайшем пути имеет следующие варианты.

1) Задача о кратчайшем пути в заданный пункт назначения. Требуется найти кратчайший путь в вершину назначения $t$, который начинается в каждой из вершин графа (кроме $t$).
2) Задача о кратчайшем пути между заданной парой вершин. Требуется найти кратчайший путь из вершины $u$ в вершину $v$.
3) Задача о кратчайшем пути между всеми парами вершин. Требуется найти кратчайший путь из каждой вершины $u$ в каждую вершину $v$.

Задача поиска кратчайшего пути в глобальном графе при планировании движения РТК относится ко второй постановке задачи о кратчайшем пути. Как показывают существующие исследования, для её решения целесообразно использовать алгоритм Беллмана-Форда или алгоритм Ли поиска кратчайшего пути.

### 1.1. Последовательный алгоритм Беллмана-Форда

В своей оригинальной трактовке алгоритм Беллмана-Форда позволяет найти длину кратчайшего пути во взвешенном графе, причём ребра графа могут иметь отрицательный вес.

Исходными данными для алгоритма являются: взвешенный граф $G = (V, E)$ (граф может быть как ориентированным, так и неориентированным), где $V$ – непустое множество вершин (узлов) графа, $E$ – множество рёбер графа; выделенная вершина $s \in V$ графа. Ниже приведён псевдокод алгоритма Беллмана-Форда, в котором $w$ – весовая функция и, соответственно, $w(u, v)$ – вес ребра $(u, v) \in E$, $d$ – массив, содержащий расстояния от вершины $s \in V$ до любой другой вершины графа $G$.

**Алгоритм 1**: Беллман-Форд $(\mathrm{G}(\mathrm{V}, \mathrm{E}), \mathrm{s}, d)$

| | | |
|---|---|---|
| 1: | d[v] ← +∞ для всех вершин v ∈ V | ▷Инициализация расстояний от вершины *s* до любой другой вершины максимально возможным значением |
| 2: | $d[s] \leftarrow 0$ | |
| 3: | $i \leftarrow 1$ | |
| 4: | **ЦИКЛ ПОКА** $i < \lvert V \rvert$ | |
| 5: | **ЦИКЛ** $(u, v) \in E$ | |
| 6: | **ЕСЛИ** $d[v] > d[u] + w(u, v)$ **ТО** | ▷Если расстояние от *s* до *v* больше суммы расстояния от s до *u* и веса ребра *(u,v)*, то… |
| 7: | $d[v] \leftarrow d[u] + w(u, v)$ | ▷…присваиваем расстоянию от *s* до *v* значение суммы расстояния от *s* до *u* и веса ребра *(u,v)* |
| 8: | **ВСЕ ЕСЛИ** | |
| 9: | **ВСЕ ЦИКЛ** | |
| 10: | $i \leftarrow i + 1$ | |
| 11: | **ВСЕ ЦИКЛ ПОКА** | |
| 12: | **РЕЗУЛЬТАТ** ← $d$ | ▷В *d* получили значения длины кратчайшего пути от *s* до каждой вершины *G* |

На каждой итерации $j$ алгоритм Беллмана-Форда позволяет определить кратчайший путь из начальной вершины $s \in V$ до любой другой вершины $v \in V$ $d[v]$ путём

прибавления к определенному на предыдущей итерации $j-1$ кратчайшему пути $d[u]$ до вершины $u \in V$ такой, что $\exists(u,v) \in E$, веса $w(u,v)$ ребра $(u,v) \in E$ только в том случае, если определённое на данный момент расстояние от начальной вершины до вершины $v \in V$ $d[v]$ превышает сумму кратчайшего пути $d[u]$ до уже рассмотренной предшествующей вершины $u \in V$ с весом ребра $(u,v) \in E$. Результатом выполнения алгоритма Беллмана-Форда станет массив $d$ кратчайших расстояний от выделенной вершины $s$ до всех остальных вершин графа $G$, что позволит определить значение кратчайшего пути между двумя вершинами графа $s \in V$ и некоторой $t \in V$, но не сам кратчайший путь как последовательность вершин. Для определения кратчайшего пути – последовательности вершин из $s$ в $t$ – необходимо модифицировать алгоритм Беллмана-Форда, добавив в него дополнительную структуру данных для хранения информации о том, какая вершина графа $G$ является предшествующей для каждой вершины $v \in V$ в кратчайшем пути, сформированном на основе алгоритма Беллмана-Форда.

### 1.2. Модифицированный последовательный алгоритм Беллмана-Форда (для определения кратчайшего пути в графе)

Номер вершины, предшествующей данной вершине в кратчайшем пути, будем хранить в массиве $r$. $t \in V$ – конечная вершина (при формулировке задачи поиска кратчайшего пути как нахождения кратчайшего пути из вершины $s$ в вершину $t$); $path$ - последовательность (список) вершин графа $G$, принадлежащих кратчайшему пути из $s$ в $t$. Ниже представлен модифицированный алгоритм Беллмана-Форда.

**Алгоритм 2**: Модифицированный Беллман-Форд (G(V, E), s, $t, d, r, path$)

| | | |
|---|---|---|
| 1: | d[v] ← +∞ для всех вершин v ∈ V | ▷Инициализация расстояний от вершины *s* до любой другой вершины максимально возможным значением |
| 2: | $d[s] \leftarrow 0$ | |
| 3: | $i \leftarrow 1$ | |
| 4: | **ЦИКЛ ПОКА** $i < \|V\|$ | |
| 5: | **ЦИКЛ** $(u,v) \in E$ | |
| 6: | **ЕСЛИ** $d[v] > d[u] + w(u,v)$ **ТО** | ▷Если расстояние от *s* до *v* больше суммы расстояния от s до *u* и веса ребра *(u,v)*, то… |
| 7: | $d[v] \leftarrow d[u] + w(u,v)$ | ▷…присваиваем расстоянию от *s* до *v* значение суммы расстояния от *s* до *u* и веса ребра *(u,v)* |
| 8: | $r[v] \leftarrow u$ | ▷Сохранение вершины, предшествующей обрабатываемой в кратчайшем пути |
| 9: | **ВСЕ ЕСЛИ** | |
| 10: | **ВСЕ ЦИКЛ** | |
| 11: | $i \leftarrow i + 1$ | |
| 12: | **ВСЕ ЦИКЛ ПОКА** | |
| 13: | $j \leftarrow t$ | ▷Начинаем просмотр массива предшествующих вершин с конечной вершины |
| 14: | $path \leftarrow t$ | ▷Инициализируем список вершин кратчайшего пути конечной вершиной |

**Алгоритм 2**: Модифицированный Беллман-Форд $(\mathrm{G(V,E)}, \mathrm{s}, t, d, r, path)$

| | | |
|---|---|---|
| 15: | **ЦИКЛ ПОКА** $j \neq s$ | ▷Пока не добрались до начальной вершины… |
| 16: | **ДОБАВИТЬ**$(path, r[j])$ | ▷…добавляем к списку вершин кратчайшего пути вершину из массива предшествующих вершин |
| 17: | $j \leftarrow r[j]$ | |
| 18: | **ВСЕ ЦИКЛ ПОКА** | |
| 19: | **ДОБАВИТЬ**$(path, s)$ | ▷Добавляем к списку вершин кратчайшего пути начальную вершину и получаем в списке $path$ перевёрнутый (от конечной вершины к начальной) кратчайший путь |
| 20: | **ПЕРЕВЕРНУТЬ**$(path)$ | ▷Переворачиваем кратчайший путь так, чтобы он представлял собой путь от начальной вершины к конечной |
| 21: | **РЕЗУЛЬТАТ** $\leftarrow d, path$ | ▷В $d$ получили значения длины кратчайшего пути от $s$ до каждой вершины $G$, в $path$ получили последовательность вершин кратчайшего пути |

Реализация процедур добавления вершины в список (**ДОБАВИТЬ**(*список, вершина*)) и переворота списка (**ПЕРЕВЕРНУТЬ**(*список*)) тривиальна. Если на вход приведённого выше алгоритма подать глобальный граф $G$, полученный из плана местности для целей планирования передвижения РКД, а также начальное положение РКД $s$ и желаемое конечное положение РКД $t$, то в списке $path$ получим кратчайший путь из начального положения РКД в конечное.

Для целей дальнейшего исследования воспользуемся модифицированной версией последовательного алгоритма Беллмана-Форда.

### 1.3. Последовательный алгоритм Ли

Алгоритм Ли осуществляет поиск кратчайшего пути на планарном графе. Преимущественно данный алгоритм используется для трассировки печатных плат, но также он широко применяется и при построении кратчайшего пути по плану местности, что подтверждает его полезность для решения задач планирования движения РКД.

Поскольку алгоритм Ли используется для невзвешенных планарных графов, то у разработчиков РКД возникает необходимость привести взвешенный глобальный граф к невзвешенному виду. Данное приведение может быть выполнено заменой каждого ребра $(u, v) \in G$ с весом ребра $c$ на связанную цепочку вершин с количеством вершин, прямо пропорциональным исходному весу ребра $c$. В частном случае число введённых вместо ребра $(u, v)$ вершин (рёбер) может быть равно весу ребра $c$, тогда получим $(u, v) \leftarrow (u, u_1) \cup \bigcup_{i=1}^{c-1}(u_i, u_{i+1}) \cup (u_c, v)$ (соответственно $(u, v) \leftarrow \bigcup_{i=1}^{c}(u_i, u_{i+1})$). На рис. 3 представлен пример вариантов перехода от взвешенной дуги (ребра) к невзвешенному представлению.

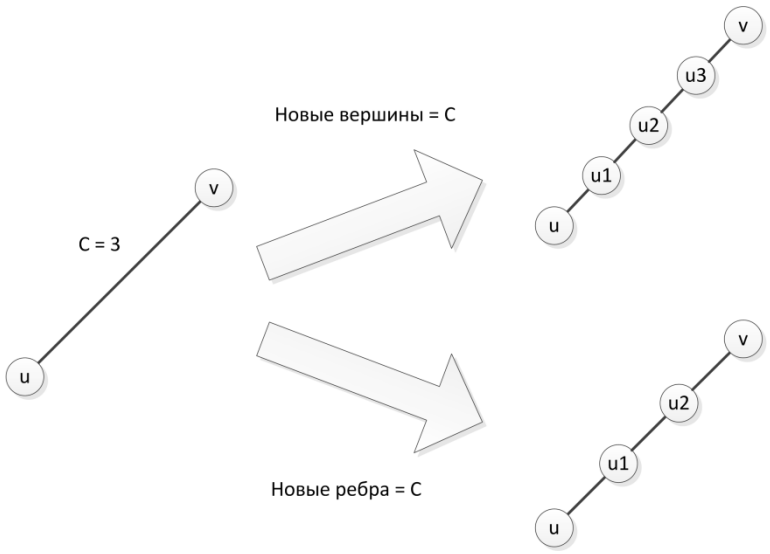



**Рис. 3.** Пример перехода от взвешенного представления к невзвешенному

Входными данными для алгоритма Ли является невзвешенный глобальный граф $G(V, E)$, а также начальная вершина пути в графе $s \in G$ и конечная вершина $t \in G$. Массив $d$ хранит числовые значения для каждой помеченной ячейки. Для непомеченных ячеек по соглашению устанавливается минимально возможное значение (отрицательное) $+\infty$, чтобы отличить их от помеченных ячеек.

**Алгоритм 3**: Ли $(\mathrm{G}(\mathrm{V}, \mathrm{E}), \mathrm{s}, t, d, path)$

1: $\mathrm{d}[\mathrm{v}] \leftarrow -\infty$ для всех вершин $\mathrm{v} \in \mathrm{V}$ ▷Инициализация числовых значений для непомеченных вершин
2: $d[s] \leftarrow 0$ ▷Инициализация для начальной вершины
3: $D \leftarrow 0$
4: **ЦИКЛ ПОКА** $d[t] = -\infty$ **И** $\exists d[v] = -\infty \forall v \in V$
5: **ЦИКЛ** $d[v] = D$
6: **ЦИКЛ** $u \in Adj(v)$ **И** $d[u] = -\infty$ ▷Найти все непомеченные смежные вершины и пометить их
7: $d[u] \leftarrow D + 1$
8: **ВСЕ ЦИКЛ**
9: **ВСЕ ЦИКЛ**
10: $D \leftarrow D + 1$
11: **ВСЕ ЦИКЛ ПОКА**
12: **ЕСЛИ** $d[t] \neq -\infty$ **ТО**
13: $cur \leftarrow t$
14: $path \leftarrow t$
15: **ЦИКЛ ПОКА** $cur \neq s$
16: $cur \leftarrow v \in Adj[cur]$ **И** $d[v] = d[cur] - 1$ ▷Найти вершину, помеченную значением на единицу меньшим текущего, и добавить её к пути
17: **ДОБАВИТЬ**$(path, cur)$
18: **ВСЕ ЦИКЛ ПОКА**
19: **ПЕРЕВЕРНУТЬ**$(path)$

**Алгоритм 3**: Ли $(\mathrm{G}(\mathrm{V},\mathrm{E}),\mathrm{s},t,d,path)$

20: **РЕЗУЛЬТАТ** $\leftarrow path$ ▷Путь найден
21: **ИНАЧЕ**
22: **РЕЗУЛЬТАТ**$\leftarrow \emptyset$ ▷Путь не найден

По своей сути алгоритм Ли определения кратчайшего пути между двумя вершинами в невзвешенном планарном графе является двухпроходовым: во время прямого прохода при помощи описанного автором «распространения волн» определяется и сохраняется численная метрика удалённости каждой вершины графа от начальной вершины $s \in G$; во время обратного прохода выстраивается цепочка – кратчайший путь – от конечной вершины $t \in G$ к начальной вершине $s$. Корректный результат выполнения алгоритма гарантируется благодаря тому, что для каждой вершины (на каждом шаге) «волна» (инкремент значений для соседних вершин) распространяется лишь на вершины, расположенные на расстоянии одного ребра от текущей.

### 1.4. Достоинства и недостатки алгоритмов Ли и Беллмана-Форда

Алгоритмы Беллмана-Форда и Ли обладают как преимуществами, так и недостатками (табл. 1), ставящими использование того или иного алгоритма для решения прикладных задач планирования движения РТК на местности в зависимость от условий решения задачи планирования движения робототехнического комплекса.

**Таблица 1.** Достоинства и недостатки алгоритмов Беллмана-Форда и Ли

| Алгоритм | Достоинства | Недостатки |
|---|---|---|
| Беллмана-Форда | • Хорошо распараллеливается<br>• Просто реализуется<br>• Не требует ресурсов памяти<br>• Требуется информация только о соседней вершине<br>• Часто заканчивается раньше $N-1$ итерации | • В худшем случае количество операций $\sim N^3$ |
| Ли | • Просто реализуется<br>• Можно найти трассу для любого плана местности и с любым количеством запретных элементов | • Сложный для распараллеливания процесс распространения волны<br>• Необходимость большого количества памяти для построения трассы |

## 2. Алгоритмы поиска кратчайшего пути в графе для выполнения в режиме сопроцессора ЭВМ МКОД

Используемая в настоящей работе методика модификации алгоритмов для реализации возможности их выполнения на ЭВМ МКОД приведена в [9].

## 2.1. Модифицированный алгоритм Беллмана-Форда для режима сопроцессора ЭВМ МКОД

*На первом этапе* распараллеливания модифицированного алгоритма Беллмана-Форда для ЭВМ МКОД необходимо определить то, как будут представлены графы и массивы в виде структур данных для хранения в памяти структур ЭВМ МКОД. Основными информационными моделями данного алгоритма являются взвешенный граф $G(V, E)$ и список вершин графа (результат) $path$. Вспомогательными информационными моделями являются массив длин кратчайших путей $d$ из вершины $s$ до всех остальных вершин графа $G$ и массив $r$, хранящий для каждой вершины графа номер вершины, предшествующей данной вершине в кратчайшем пути. Для целей распараллеливания модифицированного алгоритма Беллмана-Форда достаточно трёх структур данных, одна из которых будет включать в себя информацию из вспомогательной информационной модели (массива длин кратчайших путей).

Упомянутая структура является представлением графа $G(V, E)$. Приведённое в табл. 2 представление графа было выбрано в связи с тем, что при выполнении алгоритма требуется искать инцидентные вершины $v \in Adj[u]$ и модифицировать длины кратчайших путей от начальной вершины до остальных вершин графа. Поле ключа G.KEY хранит номер вершины $u$ и порядковый номер ребра, инцидентного данной вершине. При этом отдельно выделяются записи вида $u, 0$ в поле G.KEY, содержащие количество рёбер графа, инцидентных данной вершине, и длину кратчайшего пути из начальной вершины в заданную вершину $u$. Поле данных G.DATA хранит номер $v_i, i = \overline{1: count(u)}$инцидентной ребру с данным порядковым номером вершины (не $u$) и вес ребра $c_i, i = \overline{1: count(u)}$, инцидентного вершине $u$, чей порядковый номер представлен в соответствующем поле G.KEY.

**Таблица 2.** Совместное представление части графа $G(V, E)$ и массива $d$

| G.KEY | G.DATA |
|---|---|
| $u, 0$ | $count, d[u]$ |
| $u, 1$ | $v_1, c_1$ |
| … | … |
| $u, count$ | $v_{count}, c_{count}$ |

Представление массива $r$ может быть реализовано в виде, представленном в табл. 3. Поле ключа r.KEY хранит номер каждой вершины графа $G$, а соответствующее поле данных r.DATA хранит номер вершины графа $G$, предшествующей вершине-ключу в кратчайшем пути от начальной вершины, найденному по алгоритму Беллмана-Форда.

**Таблица 3.** Пример представления массива $r$

| r.KEY | r.DATA |
|---|---|
| $s$ | $\emptyset$ |
| $u_1$ | $s$ |
| $u_2$ | $s$ |
| $u_3$ | $u_1$ |
| … | … |
| $u_{\|V\|-1}$ | $u_3$ |

Представление списка $path$ может быть реализовано в виде, представленном в табл. 4. Поле ключа path.KEY содержит порядковый номер элемента в списке, а поле данных path.DATA содержит вершину графа, номер которой указан в элементе списка.

**Таблица 4.** Пример представления списка $path$

| path.KEY | path.DATA |
|---|---|
| 0 | $s$ |
| 1 | $u_1$ |
| … | … |
| $N-1$ | $t$ |

В ходе выполнения *второго этапа методики*, алгоритм должен быть представлен в базисе операций над структурами данных. Для МКОД системы было выделено 15 команд обработки множеств и структур данных [9]. Модифицированный алгоритм Беллмана-Форда может быть представлен в базисе этих операций над структурами данных. В частности, для этого потребуются такие операции, как: SEARCH (поиск), INSERT (добавление), POWER (мощность), DELSTR (удаление структуры), JT (переход по тегу). К примеру, вес некоторого ребра графа $G$ может быть получен путём последовательного выполнения операции поиска вершины $u$ и осуществления доступа к полученной структуре данных G.DATA. Базовые операции над структурами МКОД приведены непосредственно в псевдокоде ЦП в режиме сопроцессора, получаемого на *третьем этапе* выполнения методики. Ниже представлен псевдокод для модифицированного алгоритма Беллмана-Форда для ЦП в режиме сопроцессора.

**Алгоритм 4**: Модифицированный Беллман-Форд $(\mathrm{G}(\mathrm{V},\mathrm{E}),\mathrm{s},t,r,path)$ / псевдокод ЦП в режиме сопроцессора

```
1:  N ← POWER(V)
2:  i ← 0
3:  ЦИКЛ ПОКА i < N
4:      dt ← SEARCH(G, [i, 0])                ▷Инициализация расстояний от
5:      INSERT(G, [i, 0], [dt.DATA.count, +∞]) вершины s до любой другой вершины
6:      i ← i + 1                             максимально возможным значением
```

**Алгоритм 4**: Модифицированный Беллман-Форд $(\mathrm{G}(\mathrm{V},\mathrm{E}),\mathrm{s},t,r,path)$ / псевдокод ЦП в режиме сопроцессора

7: **ВСЕ ЦИКЛ ПОКА**
8: $INSERT(G,[s,0],[dt.DATA.count,0])$
9: $INSERT(r,[s],\emptyset)$
10: $i \leftarrow 0$
11: **ЦИКЛ ПОКА** $i < N$
12: $dt \leftarrow SEARCH(G,[i,0])$
13: $u\_index \leftarrow i$
14: $Num\_vert\_u \leftarrow dt.DATA.count$
15: $j \leftarrow 1$
16: **ЦИКЛ ПОКА** $j \leq Num\_vert\_u$
17: $dt\_next\_in \leftarrow SEARCH(G,[i,j])$
18: $v\_index \leftarrow dt\_next\_in.DATA.id$
19: $dt\_next \leftarrow SEARCH(G,[v\_index,0])$
20: **ЕСЛИ** $dt\_next.DATA.d > dt.DATA.d + dt\_next\_in.DATA.c$ **ТО**
21: $INSERT(G,[v\_index,0],[dt\_next.DATA.count, dt.DATA.d + dt\_next\_in.DATA.c)$
22: $INSERT(r,[v\_index],[u\_index])$
23: **ВСЕ ЕСЛИ**
24: $j \leftarrow j + 1$
25: **ВСЕ ЦИКЛ ПОКА**
26: $i \leftarrow i + 1$
27: **ВСЕ ЦИКЛ ПОКА**
28: $i \leftarrow 1$
29: $j \leftarrow t$
30: **ЦИКЛ ПОКА** $j \neq \emptyset$
31: $INSERT(path,[i],[j])$
32: $i \leftarrow i + 1$
33: $j \leftarrow SEARCH(r,[j])$
34: **ВСЕ ЦИКЛ ПОКА**
35: $DELSTR(r)$
36: **РЕЗУЛЬТАТ←ПЕРЕВЕРНУТЬ**$(path)$

▷Если расстояние от *s* до *v* больше суммы расстояния от s до *u* и веса ребра *(u,v)*, то присваиваем расстоянию от *s* до *v* значение суммы расстояния от *s* до *u* и веса ребра *(u,v)*

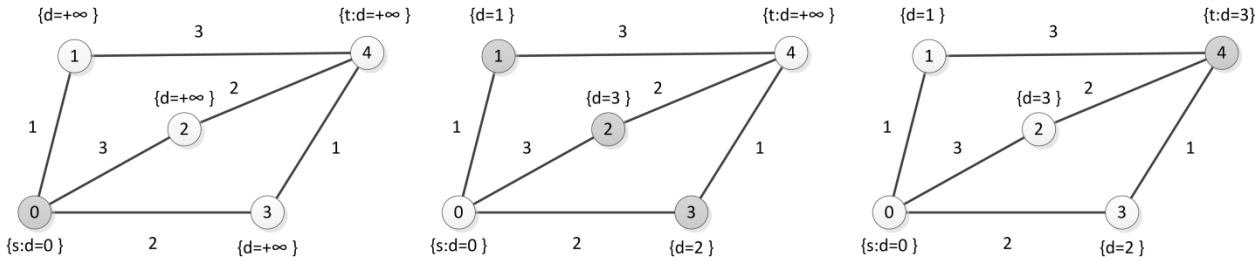
▷Сохранение вершины, предшествующей обрабатываемой в кратчайшем пути

▷Добавляем в список вершины кратчайшего пути в обратном порядке

▷Удалили ненужную структуру

▷Получили последовательность вершин кратчайшего пути

При появлении команд процессора обработки структур псевдокод алгоритма несколько усложнился, хотя в целом сохранил свою исходную структуру. Важно и то, что алгоритм с командами СП предоставляет такой же результат работы, что и обычный модифицированный алгоритм Беллмана-Форда. Чтобы лучше показать работу алгоритма в базисе команд СП, на рис. 5 приведён пример работы алгоритма 4.




| G.KEY | G.DATA |
|---|---|
| 0,0 | 3,0 |
| 0,1 | 1,1 |
| 0,2 | 2,3 |
| 0,3 | 3,2 |
| 1,0 | 2, +∞ |
| 1,1 | 0,1 |
| 1,2 | 4,3 |
| 2,0 | 2, +∞ |
| 2,1 | 0,3 |
| 2,2 | 4,2 |
| 3,0 | 2, +∞ |
| 3,1 | 0,2 |
| 3,2 | 4,1 |
| 4,0 | 3, +∞ |
| 4,1 | 1,3 |
| 4,2 | 2,2 |
| 4,3 | 3,1 |

| r.KEY | r.DATA |
|---|---|
| 0 | ∅ |
| 1 | ∅ |
| 2 | ∅ |
| 3 | ∅ |
| 4 | ∅ |

| G.KEY | G.DATA |
|---|---|
| 0,0 | 3,0 |
| 0,1 | 1,1 |
| 0,2 | 2,3 |
| 0,3 | 3,2 |
| 1,0 | 2,1 |
| 1,1 | 0,1 |
| 1,2 | 4,3 |
| 2,0 | 2,3 |
| 2,1 | 0,3 |
| 2,2 | 4,2 |
| 3,0 | 2,2 |
| 3,1 | 0,2 |
| 3,2 | 4,1 |
| 4,0 | 3, +∞ |
| 4,1 | 1,3 |
| 4,2 | 2,2 |
| 4,3 | 3,1 |

| r.KEY | r.DATA |
|---|---|
| 0 | ∅ |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | ∅ |

| G.KEY | G.DATA |
|---|---|
| 0,0 | 3,0 |
| 0,1 | 1,1 |
| 0,2 | 2,3 |
| 0,3 | 3,2 |
| 1,0 | 2,1 |
| 1,1 | 0,1 |
| 1,2 | 4,3 |
| 2,0 | 2,3 |
| 2,1 | 0,3 |
| 2,2 | 4,2 |
| 3,0 | 2,2 |
| 3,1 | 0,2 |
| 3,2 | 4,1 |
| 4,0 | 3,3 |
| 4,1 | 1,3 |
| 4,2 | 2,2 |
| 4,3 | 3,1 |

| r.KEY | r.DATA |
|---|---|
| 0 | ∅ |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 3 |

**Рис. 4.** Пример работы модифицированного алгоритма Беллмана-Форда в базисе команд СП

В табл. 5 представлен результат работы алгоритма на представленном примере, сохранённый в структуре *path*.

**Таблица 5.** Результат работы модифицированного алгоритма Беллмана-Форда

| path.KEY | path.DATA |
|---|---|
| 0 | 4 |
| 1 | 3 |
| 2 | 0 |

Таким образом, для приведённого примера получен кратчайший путь $\{s: 0\} \rightarrow \{3\} \rightarrow \{t: 4\}$ длиной 3.

### 2.2. Алгоритм Ли для режима сопроцессора ЭВМ МКОД

На первом этапе определим представление информационных моделей, используемых в алгоритме Ли, в виде структур данных. В качестве основных

информационных моделей в алгоритме Ли используются глобальный граф $G(V,E)$ с невзвешенными рёбрами, а также список вершин кратчайшего пути $path$. Помимо указанных информационных моделей, используется вспомогательная модель – массив $d$, хранящий числовые значения для каждой помеченной при выполнении алгоритма Ли вершины. Для целей модификации алгоритма Ли для режима сопроцессора при выборе структур данных целесообразно объединить представление графа $G$ и представление массива $d$ по аналогии с тем, как это было сделано для алгоритма Беллмана-Форда. Совместное представление графа и массива приведено в табл. 6. Основным отличием является отсутствие в поле G.DATA веса ребра.

**Таблица 6.** Совместное представление части графа $G(V,E)$ и массива $d$

| G.KEY | G.DATA |
|---|---|
| $u, 0$ | $count, d[u]$ |
| $u, 1$ | $v_1$ |
| … | … |
| $u, count$ | $v_{count}$ |

Представление основной информационной модели списка вершин кратчайшего пути графа $path$ не отличается от аналогичной модели для модифицированного алгоритма Беллмана-Форда.

Для упрощения процедуры поиска вершины по числовому значению помеченной вершины (см. строку 5 алгоритма 3) целесообразно ввести вспомогательную структуру, в которой ключом является значение числового значения группы помеченных вершин (одним и тем же значением может быть помечено несколько вершин) и порядковый номер этой вершины в списке помеченных этим значением. Представление структуры приведено в табл. 7.

**Таблица 7.** Представление вспомогательной структуры данных для поиска помеченных вершин по численному значению *D*

| DS.KEY | DS.DATA |
|---|---|
| 0 | $s$ |
| 1,0 | $count_1$ |
| 1,1 | $v_1$ |
| 1,2 | $v_2$ |
| … | … |
| $D_N, count$ | $v$ |

Псевдокод для алгоритма Ли для ЦП в режиме сопроцессора с применением операций из базиса операций над структурами данных приведён ниже.

**Алгоритм 5**: Ли $(\mathrm{G(V,E)}, \mathrm{s}, t, DS, path)$ / псевдокод ЦП в режиме сопроцессора

1: $N \leftarrow POWER(V)$
2: $i \leftarrow 0$
3: **ЦИКЛ ПОКА** $i < N$
4: $dt \leftarrow SEARCH(G, [i, 0])$ ▷Инициализация помеченных вершин

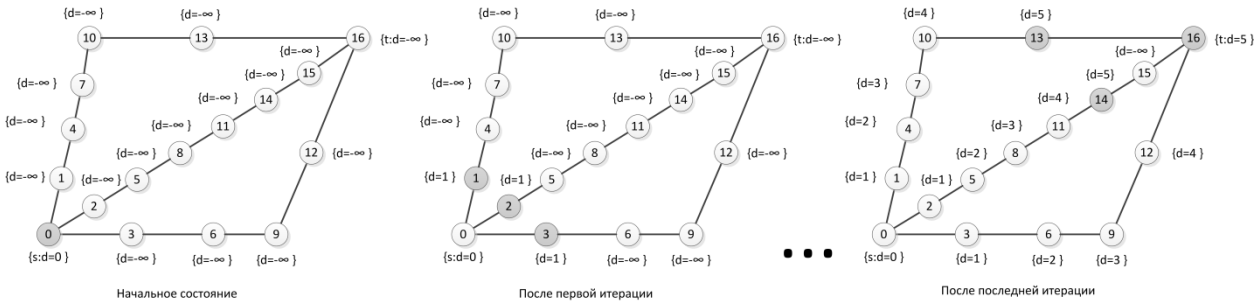
**Алгоритм 5**: Ли (G(V, E), s, $t$, $DS$, $path$) / псевдокод ЦП в режиме сопроцессора

```
5:      INSERT(G, [i, 0], [dt.DATA.count, −∞])          минимально возможным значением
6:      i ← i + 1
7:  ВСЕ ЦИКЛ ПОКА
8:  INSERT(G, [s, 0], [dt.count, 0])
9:  pom ← 0                                  ▷Счётчик числа помеченных вершин в графе
10: dt ← SEARCH(G, [t, 0])
11: D ← 0                                    ▷Значение, которым будут помечены вершины
12: ЦИКЛ ПОКА (dt.DATA.d = −∞) И (pom < N)
13:     dt_ind_DS ← SEARCH(DS, [D, 0])
14:     iter ← dt_ind_DS.DATA.count
15:     ЦИКЛ ПОКА iter > 0
16:         v ← SEARCH(DS, [D, iter])            ▷Ищем вершину по значению D
17:         dt_fnd_in_G ← SEARCH(G, [v, 0])      ▷Ищем информацию по найденной вершине в
18:         i ← 1                                                              G
19:         cnt ← 0                              ▷Счётчик вершин с одинаковым значением D
20:         ЦИКЛ ПОКА i ≤ dt_fnd_in_G.DATA.count
21:             dtG ← SEARCH(G, [v, i])          ▷Находим смежную вершину и определяем,
22:             dtAdj ← SEARCH(G, [dtG.DATA.id, 0])   помеченная она или нет
23:             ЕСЛИ dtAdj.DATA.d = −∞ ТО
24:                 pom ← pom + 1                ▷Увеличилось количество помеченных вершин
25:                 cnt ← cnt + 1                ▷Увеличилось количество вершин для D
26:                 INSERT(G, [dtAdj.DATA.id, 0],   ▷Помечаем вершину в графе G
    [dtAdj.DATA.count, D + 1])
27:                 INSERT(DS, [D + 1, i], [dtG.DATA.id])   ▷Вставляем информацию о помеченной
                                                                          вершине в DS
28:             ВСЕ ЕСЛИ
29:             i ← i + 1
30:         ВСЕ ЦИКЛ ПОКА
31:         INSERT(DS, [D + 1,0], [cnt])         ▷Вставляем заглавную вершину в индекс DS
32:         D ← D + 1                            ▷Готовимся помечать следующим значением
33:         iter ← iter − 1
34:     ВСЕ ЦИКЛ ПОКА
35:     dt ← SEARCH(G, [t, 0])
36: ВСЕ ЦИКЛ ПОКА
37: DELSTR(DS)                               ▷Удаляем ненужную структуру
38: ЕСЛИ dt.DATA.d ≠ −∞ ТО
39:     cur ← t
40:     key ← 0                              ▷Ключ в структуре path
41:     INSERT(path, [key], [t])             ▷Начинаем с конца
42:     ЦИКЛ ПОКА cur ≠ s
43:         dtcur ← SEARCH(G, [cur, 0])
44:         curD ← dtcur.DATA.d
45:         i ← 1
46:         ЦИКЛ ПОКА i ≤ dtcur.DATA.count
47:             dtAdj ← SEARCH(G, [cur, i])
48:             num ← dtAdj.DATA.id
49:             dtAdjMain ← SEARCH(G, [num, 0])  ▷Величина D для помеченной смежной
```

**Алгоритм 5**: Ли (G(V, E), s, $t$, $DS$, $path$) / псевдокод ЦП в режиме сопроцессора

```
50:          D_Adj ← dtAdjMain.DATA.d                          вершины
51:          ЕСЛИ D_Adj = curD − 1 ТО
52:             key ← key + 1
53:             INSERT(path, [key], [num])          ▷Вставляем в кратчайший путь следующую
54:             cur ← num                                      вершину
55:          ВСЕ ЕСЛИ
56:          i ← i + 1
57:       ВСЕ ЦИКЛ ПОКА
58:    ВСЕ ЦИКЛ ПОКА
59:    РЕЗУЛЬТАТ ← ПЕРЕВЕРНУТЬ(path)          ▷Результат – кратчайший путь из s в t
60: ИНАЧЕ
61:    РЕЗУЛЬТАТ ← ∅                                          ▷Путь не найден
62: ВСЕ ЕСЛИ
```

Работа алгоритма Ли в базисе команд СП показана на рис. 5 на примере небольшого графа.



Начальное состояние

| G.KEY | G.DATA |
|---|---|
| 0,0 | 3,0 |
| 0,1 | 1 |
| 0,2 | 2 |
| 0,3 | 3 |
| 1,0 | 2, −∞ |
| 1,1 | 4 |
| 1,2 | 0 |
| ... | ... |
| 15,0 | 2, −∞ |
| 15,1 | 16 |
| 15,2 | 14 |
| 16,0 | 3, −∞ |
| 16,1 | 13 |
| 16,2 | 15 |
| 16,3 | 12 |

| DS.KEY | DS.DATA |
|---|---|
| 0 | 0 |

После первой итерации

| G.KEY | G.DATA |
|---|---|
| 0,0 | 3,0 |
| 0,1 | 1 |
| 0,2 | 2 |
| 0,3 | 3 |
| 1,0 | 2,1 |
| 1,1 | 4 |
| 1,2 | 0 |
| ... | ... |
| 15,0 | 2, −∞ |
| 15,1 | 16 |
| 15,2 | 14 |
| 16,0 | 3, −∞ |
| 16,1 | 13 |
| 16,2 | 15 |
| 16,3 | 12 |

| DS.KEY | DS.DATA |
|---|---|
| 0 | 0 |
| 1,0 | 3 |
| 1,1 | 1 |
| 1,2 | 2 |
| 1,3 | 3 |

После последней итерации

| G.KEY | G.DATA |
|---|---|
| 0,0 | 3,0 |
| 0,1 | 1 |
| 0,2 | 2 |
| 0,3 | 3 |
| 1,0 | 2,1 |
| 1,1 | 4 |
| 1,2 | 0 |
| ... | ... |
| 15,0 | 2, −∞ |
| 15,1 | 16 |
| 15,2 | 14 |
| 16,0 | 3,5 |
| 16,1 | 13 |
| 16,2 | 15 |
| 16,3 | 12 |

| DS.KEY | DS.DATA |
|---|---|
| 0 | 0 |
| 1,0 | 3 |
| 1,1 | 1 |
| 1,2 | 2 |
| 1,3 | 3 |
| ... | ... |
| 5,0 | 3 |
| 5,1 | 13 |
| 5,2 | 14 |
| 5,3 | 16 |

**Рис. 5.** Пример работы алгоритма Ли в базисе команд СП

В табл. 8 представлен результат работы алгоритма на приведённом ранее примере, сохранённый в структуре *path*.

**Таблица 8.** Результат работы алгоритма Ли

| path.KEY | path.DATA |
|---|---|
| 0 | 16 |
| 1 | 12 |
| 2 | 9 |
| 3 | 6 |
| 4 | 3 |
| 5 | 0 |

Для примера мы получили кратчайший путь $\{s\colon 0\} \to \{3\} \to \{6\} \to \{9\} \to \{12\} \to \{t\colon 16\}$, содержащий 6 вершин.

## 3. Алгоритмы поиска кратчайшего пути в режиме МКОД

Представление алгоритмов в режиме МКОД позволяет отделить поток команд для обработки структур данных от потока команд, обрабатывающих информационную компоненту структур данных. Получение алгоритмов в режиме МКОД с минимально возможным количеством пересылок данных между ЦП и СП является целью процедуры распараллеливания, поскольку позволяет выполнять алгоритмы с высокой скоростью, снижая временные издержки, возникающие при пересылке данных между процессором структур и центральным процессором.

### 3.1. Модифицированный алгоритм Беллмана-Форда в режиме МКОД

Представление модифицированного алгоритма Беллмана-Форда в режиме МКОД приведено ниже. Важно отметить высокую интенсивность использования алгоритмом команд базиса обработки структур данных, что, скорее всего, влечёт за собой дополнительные возможности по оптимизации кода путём перестановки команд для уменьшения количества пересылок данных между ЦП и СП.

**Алгоритм 6**: Модифицированный Беллман-Форд (G(V, E), s, $t, r, path$) / псевдокод в режиме МКОД

| | **Поток ЦП** | **▷Поток СП** |
|---|---|---|
| 1: | $N \leftarrow GET$ | $\triangleright POWER(V)$ |
| 2: | $i \leftarrow 0$ | |
| 3: | **ЦИКЛ ПОКА** $i < N$ | ▷**bf_label1:** $JT(?, bf\_label2)$ |
| 4: | $PUT(0)$ | |
| 5: | $PUT([i, 0])$ | $\triangleright SEARCH(G, ?)$ |
| 6: | $dt \leftarrow GET$ | |
| 7: | $PUT([i, 0]^*, [dt.DATA.count, +\infty])$ | $\triangleright INSERT(G, ?, ?)$ |
| 8: | $i \leftarrow i + 1$ | |
| 9: | **ВСЕ ЦИКЛ ПОКА** | $\triangleright JT(1, bf\_label1)$ |
| 10: | $PUT(1)$ | ▷**bf_label2:** |
| 11: | $PUT([s, 0], [dt.DATA.count, 0])$ | $\triangleright INSERT(G, ?, ?)$ |

**Алгоритм 6**: Модифицированный Беллман-Форд (G(V, E), s, $t, r, path$) / псевдокод в режиме МКОД

12: $i \leftarrow 0$ ▷$INSERT(r, [s], \emptyset)$
13: **ЦИКЛ ПОКА** $i < N$ ▷**bf_label3:**$JT(?, bf\_label4)$
14: $PUT(0)$
15: $PUT([i, 0])$ ▷$SEARCH(G, ?)$
16: $dt \leftarrow GET$
17: $u\_index \leftarrow i$
18: $Num\_vert\_u \leftarrow dt.DATA.count$
19: $j \leftarrow 1$
20: **ЦИКЛ ПОКА** $j \leq Num\_vert\_u$ ▷**bf_label5:**$JT(?, bf\_label6)$
21: $PUT(0)$
22: $PUT([i, j])$ ▷$SEARCH(G, ?)$
23: $dt\_next\_in \leftarrow GET$
24: $v\_index \leftarrow dt\_next\_in.DATA.id$
25: $PUT([v\_index, 0])$ ▷$SEARCH(G, ?)$
26: $dt\_next \leftarrow GET$
27: **ЕСЛИ** $dt\_next.DATA.d > dt.DATA.d + dt\_next\_in.DATA.c$ **ТО**
28: $PUT(0)$ ▷$JT(?, bf\_label7)$
29: $newnum \leftarrow dt.DATA.d + dt\_next\_in.DATA.c$
30: $PUT([v\_index^*, 0], [dt\_next.DATA.count, newnum])$ ▷$INSERT(G, ?, ?)$
31: $PUT([v\_index^*], [u\_index])$ ▷$INSERT(r, ?, ?)$
32: **ВСЕ ЕСЛИ**
33: $PUT(1)$ ▷**bf_label7:**
34: $j \leftarrow j + 1$
35: **ВСЕ ЦИКЛ ПОКА** ▷$JT(1, bf\_label5)$
36: $PUT(1)$ ▷**bf_label6:**
37: $i \leftarrow i + 1$
38: **ВСЕ ЦИКЛ ПОКА** ▷$JT(1, bf\_label3)$
39: $PUT(1)$ ▷**bf_label4:**
40: $i \leftarrow 1$
41: $j \leftarrow t$
42: **ЦИКЛ ПОКА** $j \neq \emptyset$ ▷**bf_label8:**$JT(?, bf\_label9)$
43: $PUT(0)$
44: $PUT([i], [j])$ ▷$INSERT(path, ?, ?)$
45: $PUT([j]^*)$ ▷$SEARCH(r, ?)$
46: $i \leftarrow i + 1$
47: $j \leftarrow GET$
48: **ВСЕ ЦИКЛ ПОКА** ▷$JT(1, bf\_label8)$
49: $PUT(1)$ ▷**bf_label9:**
50: **РЕЗУЛЬТАТ←ПЕРЕВЕРНУТЬ**($path$) ▷$DELSTR(r)$

Полученный алгоритм Беллмана-Форда может быть подвергнут дальнейшей оптимизации для уменьшения количества связей между потоками ЦП и СП, но для эффективности этой операции необходимо провести разработку метрики и процедуры оптимизации, основывающейся на уменьшении числа пересылок между потоками ЦП и СП.

### 3.2. Алгоритм Ли в режиме МКОД

Представление алгоритма Ли в режиме МКОД приведено ниже. Стоит отметить, что по сравнению с режимом сопроцессора СП количество строк псевдокода в алгоритме Ли в режиме МКОД выросло почти в 1,5 раза. Это увеличение количества псевдокода связано с содержательными особенностями алгоритма и, скорее всего, свидетельствует о возможностях дальнейшей оптимизации.

**Алгоритм 7**: Ли (G(V, E), s, $t, DS, path$) / псевдокод в режиме МКОД

| | **Поток ЦП** | **▷Поток СП** |
|---|---|---|
| 1: | $N \leftarrow GET$ | $\triangleright POWER(V)$ |
| 2: | $i \leftarrow 0$ | |
| 3: | **ЦИКЛ ПОКА** $i < N$ | ▷**li_label1:** $JT(?, li\_label2)$ |
| 4: | $PUT(0)$ | |
| 5: | $PUT([i, 0])$ | $\triangleright SEARCH(G, ?)$ |
| 6: | $dt \leftarrow GET$ | |
| 7: | $PUT([i, 0]^*, [dt.DATA.count, +\infty])$ | $\triangleright INSERT(G, ?, ?)$ |
| 8: | $i \leftarrow i + 1$ | |
| 9: | **ВСЕ ЦИКЛ ПОКА** | $\triangleright JT(1, li\_label1)$ |
| 10: | $PUT(1)$ | ▷**li_label2:** |
| 11: | $PUT([s, 0], [dt.DATA.count, 0])$ | $\triangleright INSERT(G, ?, ?)$ |
| 12: | $pom \leftarrow 0$ | |
| 13: | $PUT([t, 0])$ | $\triangleright SEARCH(G, ?)$ |
| 14: | $dt \leftarrow GET$ | |
| 15: | $D \leftarrow 0$ | |
| 16: | **ЦИКЛ ПОКА** $(dt.DATA.d = -\infty)$ **И** $(pom < N)$ | ▷**li_label3:** $JT(?, li\_label4)$ |
| 17: | $PUT(0)$ | |
| 18: | $PUT([D, 0])$ | $\triangleright SEARCH(DS, ?)$ |
| 19: | $dt\_ind\_DS \leftarrow GET$ | |
| 20: | $iter \leftarrow dt\_ind\_DS.DATA.count$ | |
| 21: | **ЦИКЛ ПОКА** $iter > 0$ | ▷**li_label5:** $JT(?, li\_label6)$ |
| 22: | $PUT(0)$ | |
| 23: | $PUT([D, iter])$ | $\triangleright SEARCH(DS, ?)$ |
| 24: | $v \leftarrow GET$ | |
| 25: | $PUT([v, 0])$ | $\triangleright SEARCH(G, ?)$ |
| 26: | $dt\_fnd\_in\_G \leftarrow GET$ | |
| 27: | $i \leftarrow 1$ | |
| 28: | $cnt \leftarrow 0$ | |
| 29: | **ЦИКЛ ПОКА** $i \leq dt\_fnd\_in\_G.DATA.count$ | ▷**li_label7:** $JT(?, li\_label8)$ |
| 30: | $PUT(0)$ | |
| 31: | $PUT([v, i])$ | $\triangleright SEARCH(G, ?)$ |
| 32: | $dtG \leftarrow GET$ | |
| 33: | $PUT([dtG.DATA.id, 0])$ | $\triangleright SEARCH(G, ?)$ |
| 34: | $dtAdj \leftarrow GET$ | |
| 35: | **ЕСЛИ** $dtAdj.DATA.d = -\infty$ **ТО** | |
| 36: | $PUT(0)$ | $\triangleright JT(?, li\_label9)$ |
| 37: | $pom \leftarrow pom + 1$ | |

**Алгоритм 7**: Ли (G(V, E), s, $t, DS, path$) / псевдокод в режиме МКОД

```
38:                 cnt ← cnt + 1
39:                 D1 ← D + 1
40:                 PUT([dtAdj.DATA.id, 0],  [dtAdj.DATA.count, D1])        ▷INSERT(G, ?, ?)
41:                 PUT([D1, i], [dtG.DATA.id])                             ▷INSERT(DS, ?, ?)
42:             ВСЕ ЕСЛИ
43:             PUT(1)                                                      ▷li_label9:
44:             i ← i + 1
45:         ВСЕ ЦИКЛ ПОКА                                                   ▷JT(1, li_label7)
46:         PUT(1)                                                          ▷li_label8:
47:         PUT([D1,0], [cnt])                                              ▷INSERT(DS, ?, ?)
48:         D ← D + 1
49:         iter ← iter − 1
50:     ВСЕ ЦИКЛ ПОКА                                                       ▷JT(1, li_label5)
51:     PUT(1)                                                              ▷li_label6:
52:     PUT([t, 0])                                                         ▷SEARCH(G, ?)
53:     dt ← GET
54: ВСЕ ЦИКЛ ПОКА                                                           ▷JT(1, li_label3)
55: PUT(1)                                                                  ▷li_label4:
56: ЕСЛИ dt.DATA.d ≠ −∞ ТО                                                  ▷DELSTR(DS)
57:     PUT(0)                                                              ▷JT(?, li_label10)
58:     cur ← t
59:     key ← 0
60:     PUT([key], [t])                                                     ▷INSERT(path, ?, ?)
61:     ЦИКЛ ПОКА cur ≠ s                                                   ▷li_label12:JT(?, li_label13)
62:         PUT(0)
63:         PUT([cur, 0])                                                   ▷SEARCH(G, ?)
64:         i ← 1
65:         dtcur ← GET
66:         curD ← dtcur.DATA.d
67:         ЦИКЛ ПОКА i ≤ dtcur.DATA.count                                  ▷li_label14:JT(?, li_label15)
68:             PUT(0)
69:             PUT([cur, i])                                               ▷SEARCH(G, ?)
70:             PUT([num, 0])                                               ▷SEARCH(G, ?)
71:             dtAdj ← GET
72:             dtAdjMain ← GET
73:             num ← dtAdj.DATA.id
74:             D_Adj ← dtAdjMain.DATA.d
75:             ЕСЛИ D_Adj = curD − 1 ТО
76:                 PUT(0)                                                  ▷JT(?, li_label16)
77:                 key ← key + 1
78:                 PUT([key], [num])                                       ▷INSERT(path, ?, ?)
79:                 cur ← num
80:             ВСЕ ЕСЛИ
81:             PUT(1)                                                      ▷li_label16:
82:             i ← i + 1
83:         ВСЕ ЦИКЛ ПОКА                                                   ▷JT(1, li_label14)
```

**Алгоритм 7**: Ли (G(V, E), s, $t, DS, path$) / псевдокод в режиме МКОД

```
84:        PUT(1)                                          ▷li_label15:
85:     ВСЕ ЦИКЛ ПОКА                                      ▷JT(1, li_label12)
86:     PUT(1)                                             ▷li_label13:
87:     РЕЗУЛЬТАТ ← ПЕРЕВЕРНУТЬ(path)                      ▷JT(1, li_label11)
88:  ИНАЧЕ                                                 ▷li_label10:
89:     PUT(1)
90:     РЕЗУЛЬТАТ ← ∅
91:  ВСЕ ЕСЛИ
92:                                                        ▷li_label11:
```

Для оценки эффективности преобразования алгоритмов в режиме МКОД была подсчитана статистика по количеству команд, наглядно характеризующая алгоритмы Беллмана-Форда и Ли в режиме МКОД. Статистика приведена в табл. 9. В число независимых команд обработки структур выделены те команды, которым не требуется ожидать данные от потока ЦП для выполнения действий. Также в таблице для каждого алгоритма приведено количество команд, использующих повторно полученные данные (такие команды в листинге алгоритмов отмечены символом «*») – организация вычислительного процесса с учётом возможности использования этих команд без повторной пересылки данных может уменьшить число пересылок и увеличить производительность алгоритмов на ЭВМ МКОД.

**Таблица 9.** Количество независимых команд обработки структур и множеств

| Алгоритм | Число строк псевдокода | Количество команд обработки структур | Количество независимых команд обработки структур | Доля независимых команд обработки структур | Количество команд, использующих повторно полученные данные |
|---|---|---|---|---|---|
| Модифицированный Беллмана-Форда | 50 | 13 | 3 | 23% | 4 |
| Ли | 92 | 20 | 2 | 10% | 1 |

Поскольку оба алгоритма решают одну и ту же задачу, можно сделать вывод о том, что в режиме МКОД гораздо эффективнее сможет показать себя модифицированный алгоритм Беллмана-Форда. Возможно улучшение характеристик работы обоих алгоритмов в режиме МКОД путём проведения дальнейшей оптимизации и уменьшения числа пересылок данных между СП и ЦП.

Тем не менее, данная методика определения эффективности алгоритмов в режиме МКОД несовершенна, поскольку не учитывает важные для параллельных вычислений диспозиционные характеристики команд алгоритмов: например, можно разместить команды в потоке ЦП таким образом, что несколько раз подряд СП будут отправлены данные, необходимые для выполнения зависимых команд, после чего (до того, как ЦП понадобятся результаты обработки) ЦП выполнит несколько команд.

## 4. Направления развития методики модификации алгоритмов для ЭВМ МКОД

Методика модификации алгоритмов для ЭВМ МКОД хотя и позволяет получить работоспособные алгоритмы для ЭВМ МКОД, имеет ряд ограничений, выявленных в процессе применения методики [9] к алгоритмам Беллмана-Форда и Ли поиска кратчайшего пути в графе:

- необходимость вручную определять наиболее оптимальное представление структур данных (с точки зрения соотношения объёма памяти/скорости выполнения операций);
- большие затраты времени на доработку псевдокода алгоритма до псевдокода режима сопроцессора СП;
- большие затраты времени на распараллеливание алгоритма для работы в режиме МКОД;
- высокая сложность и высокие трудозатраты оптимизации распараллеленных программ в части уменьшения зависимостей по данным между данными, обрабатываемыми ЦП, и данными, обрабатываемыми СП;
- высокая трудоёмкость распараллеливания алгоритма для работы в режиме МКОД для алгоритмов с общим числом строк псевдокода, превышающим 100;
- значительное увеличение сложности восприятия псевдокода алгоритма для работы в режиме МКОД.

Решение перечисленных проблем представляет собой отдельную научно-техническую задачу, требующую детальной проработки. Многие из представленных проблем могут быть решены при помощи разработки специальных методов и средств модификации последовательных алгоритмов, позволяющих в полностью автоматическом режиме формировать код для выполнения как в режиме сопроцессора СП, так и в режиме МКОД.

Можно выделить следующие ключевые направления развития методики модификации алгоритмов для ЭВМ МКОД:

- выбор или разработка математической модели, описывающей алгоритмы в информационно-структурной парадигме (то есть математическая модель должна разделять потоки команд над структурами данных и над информацией, записанной в эти структуры, а также показывать связи между указанными потоками), при этом крайне важно, чтобы указанная математическая модель обеспечивала наглядное представление алгоритма для возможного участия человека в процессе модификации алгоритма для режима МКОД;
- дополнение методики формализованными (для последующей реализации на ЭВМ) методами определения зависимостей между потоком команд, обрабатывающим структуры данных, и потоком команд, обрабатывающим данные в структурах;

- создание базы типовых алгоритмических конструкций для их замены на предварительно разработанные конструкции алгоритмов в режиме МКОД;
- разработка методов, позволяющих выделить необходимые для обеспечения эффективного представления алгоритма в режиме МКОД структуры данных (включая создание и обработку вспомогательных структур, если в этом имеется необходимость);
- разработка методов оптимизации использования данных ЦП и СП для снижения обмена данными между ЦП и СП;
- разработка методов и метрик, позволяющих определить целесообразность преобразования частей алгоритма в режим МКОД;
- разработка средств, позволяющих оценить эффективность обработки данных каждым потоком команд МКОД и интенсивность обмена данными между ЦП и СП.

Работа в рамках перечисленных направлениях в конечном итоге должна привести к созданию комплексного набора средств автоматического преобразования алгоритмов к формату, обрабатываемому в режиме сопроцессора СП или в режиме МКОД. Наличие подобных средств позволит ЭВМ МКОД перейти на уровень, соответствующий решению прикладных задач.

## Заключение

В статье показана практическая значимость ЭВМ МКОД для одного из направлений разработки робототехнических комплексов – планирование движения РКД. Поскольку автоматическое планирование движения РКД широко использует алгоритмы, в основе которых лежат структуры данных (графы), целесообразно преобразование этих алгоритмов в формат, применяемый для обработки данных на ЭВМ МКОД. В качестве примера применения методики модификации алгоритмов для ЭВМ МКОД были использованы алгоритмы Беллмана-Форда и Ли, применяющиеся в робототехнике для поиска кратчайшего пути между двумя точками на местности. В результате анализа процесса преобразования данных алгоритмов были выявлены ограничения существующей методики. Устранение выявленных ограничений позволит использовать ЭВМ МКОД для решения полноценных прикладных задач. Для устранения ограничений были предложены направления развития оригинальной методики преобразования алгоритмов. Предложенные направления в данный момент прорабатываются и будут представлены в будущих работах.

## Список литературы

1. Носков В.П., Рубцов И.В. Ключевые вопросы создания интеллектуальных мобильных роботов // Инженерный журнал: наука и инновации. 2013. № 3. Режим доступа: http://engjournal.ru/catalog/pribor/robot/629.html (дата обращения 01.03.2015).

2. Евсеев А.А., Носков В.П., Платонов А.К. Электронная карта в системе управления автономным движением мобильного робота // Известия Тульского государственного университета. Сер. Вычислительная техника, информационные технологии, системы управления. 2006. Т. 1, вып. 3 «Системы управления». С. 166-169.
3. Каляев А.В., Носков В.П., Чернухин Ю.В. Алгоритм управляющей структуры транспортного робота // Известия АН СССР. Техническая кибернетика. 1980. № 4. С. 64-72.
4. Cherkassky B.V., Goldberg A.V., Radzik T. Shortest paths algorithms: theory and experimental evaluation // Mathematical Programming. Ser. A. 1996. Vol. 73, no. 2. P. 129-174. DOI: 10.1016/0025-5610(95)00021-6
5. Abraham I., Fiat A., Goldberg A.V., Werneck R.F. Highway Dimension, Shortest Paths, and Provably Efficient Algorithms // Proc. of the 21 annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10). SIAM, 2010. P. 782-793.
6. Abraham I., Delling D., Goldberg A.V., Werneck R.F. A Hub-Based Labeling Algorithm for Shortest Paths on Road Networks // In: Experimental Algorithms. Proc. 10th International Symposium on Experimental Algorithms (SEA 2011). Springer Berlin Heidelberg, 2011. P. 230-241. DOI: 10.1007/978-3-642-20662-7_20
7. Thorup M. Undirected single-source shortest paths with positive integer weights in linear time // Journal of the ACM (JACM). 1999. Vol. 46, no. 3. P. 362-394. DOI: 10.1145/316542.316548
8. Попов А.Ю. Применение вычислительных систем с многими потоками команд и одним потоком данных для решения задач оптимизации // Инженерный журнал: наука и инновации. 2012. № 1. Режим доступа: http://engjournal.ru/catalog/it/hidden/80.html (дата обращения 01.03.2015).
9. Попов А.Ю. О реализации алгоритма Форда-Фалкерсона в вычислительной системе с многими потоками команд и одним потоком данных // Наука и Образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2014. № 9. С. 162-180. DOI: 10.7463/0914.0726416
10. Харари Ф. Теория графов: пер. с англ. / под ред. Г.П. Гаврилова. М.: УРСС: ЛЕНАНД, 2015. 304 с.
11. Алексеев В.Е., Таланов В.А. Графы. Модели вычислений. Структуры данных. Н. Новгород: Изд-во Нижегородского гос. ун-та, 2005. 307 с.

Science & Education of the Bauman MSTU
Electronic journal
ISSN 1994-0408

*Science and Education of the Bauman MSTU*, 2015, no. 04, pp. 189–214.

DOI: **10.7463/0415.0764268**

Received: 25.02.2015
Revised: 04.04.2015



# On the Organization of Parallel Operation of Some Algorithms for Finding the Shortest Path on a Graph on a Computer System with Multiple Instruction Stream and Single Data Stream

**Podol'skii V.E.**[1,*]

[*]v.e.podolskiy@gmail.com

[1]Bauman Moscow State Technical University, Moscow, Russia

**Keywords:** Bellman-Ford algorithm, Lee algorithm, multiple instruction stream and single data stream, MISD, structures processor

The paper considers the implementing Bellman-Ford and Lee algorithms to find the shortest graph path on a computer system with multiple instruction stream and single data stream (MISD). The MISD computer is a computer that executes commands of arithmetic-logical processing (on the CPU) and commands of structures processing (on the structures processor) in parallel on a single data stream. Transformation of sequential programs into the MISD programs is a labor intensity process because it requires a stream of the arithmetic-logical processing to be manually separated from that of the structures processing. Algorithms based on the processing of data structures (e.g., algorithms on graphs) show high performance on a MISD computer. Bellman-Ford and Lee algorithms for finding the shortest path on a graph are representatives of these algorithms. They are applied to robotics for automatic planning of the robot movement in-situ. Modification of Bellman-Ford and Lee algorithms for finding the shortest graph path in coprocessor MISD mode and the parallel MISD modification of these algorithms were first obtained in this article. Thus, this article continues a series of studies on the transformation of sequential algorithms into MISD ones (Dijkstra and Ford-Fulkerson 's algorithms) and has a pronouncedly applied nature. The article also presents the analysis results of Bellman-Ford and Lee algorithms in MISD mode. The paper formulates the basic trends of a technique for parallelization of algorithms into arithmetic-logical processing stream and structures processing stream. Among the key areas for future research, development of the mathematical approach to provide a subsequently formalized and automated process of parallelizing sequential algorithms between the CPU and structures processor is highlighted. Among the mathematical models that can be used in future studies there are graph models of algorithms (e.g., dependency graph of a program). Due to the high labor intensity of manual parallelization of sequential computer programs for MISD computer, automatic parallelization becomes a key issue for the development of a new MISD architecture. The paper lays down the foundation for the solution of this task.

## References


1. Noskov V.P., Rubtsov I.V. Key issues of intelligent mobile robots design. *Inzhenernyy zhurnal: nauka i innovatsii = Engineering Journal: Science and Innovation*, 2013, no. 3. Available at: http://engjournal.ru/catalog/pribor/robot/629.html , accessed 01.03.2015. (in Russian).
2. Evseev A.A., Noskov V.P., Platonov A.K. Electronic card in control system of autonomous mobile robot motion. *Izvestiya Tul'skogo gosudarstvennogo universiteta. Ser. Vychislitel'naya tekhnika, informatsionnye tekhnologii, sistemy upravleniya*, 2006, vol. 1, iss. 3, pp. 166-169. (in Russian).
3. Kalyaev A.V., Noskov V.P., Chernukhin Yu.V. Algorithm of control structure of transport robot. *Izvestiya AN SSSR. Tekhnicheskaya kibernetika*, 1980, no. 4, pp. 64-72. (in Russian).
4. Cherkassky B.V., Goldberg A.V., Radzik T. Shortest paths algorithms: theory and experimental evaluation. *Mathematical Programming. Ser. A*, 1996, vol. 73, no. 2, pp. 129-174. DOI: 10.1016/0025-5610(95)00021-6
5. Abraham I., Fiat A., Goldberg A.V., Werneck R.F. Highway Dimension, Shortest Paths, and Provably Efficient Algorithms. *Proc. of the 21 annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10).* SIAM, 2010, pp. 782-793.
6. Abraham I., Delling D., Goldberg A.V., Werneck R.F. A Hub-Based Labeling Algorithm for Shortest Paths on Road Networks. In: *Experimental Algorithms. Proc. 10th International Symposium on Experimental Algorithms (SEA 2011)*. Springer Berlin Heidelberg, 2011, pp. 230-241. DOI: 10.1007/978-3-642-20662-7_20
7. Thorup M. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM (JACM)*, 1999, vol. 46, no. 3, pp. 362-394. DOI: 10.1145/316542.316548
8. Popov A.Yu. Application of computer systems with multiple instruction streams and single data stream for solving optimization problems. *Inzhenernyy zhurnal: nauka i innovatsii = Engineering Journal: Science and Innovation*, 2012, no. 1. Available at: http://engjournal.ru/catalog/it/hidden/80.html , accessed 01.03.2015. (in Russian).
9. Popov A.Yu. On the implementation of the Ford-Fulkerson algorithm on the Multiple Instruction and Single Data computer system. *Nauka i obrazovanie MGTU im. N.E. Baumana = Science and Education of the Bauman MSTU*, 2014, no. 9, pp. 162-180. DOI: 10.7463/0914.0726416 (in Russian).
10. Harary F. *Graph Theory*. Addison-Wesley, Reading, MA, 1969. (Russ. ed.: Harary F. *Teoriya grafov*. Moscow, URSS: LENAND Publ., 2015. 304 p.).
11. Alekseev V.E., Talanov V.A. *Grafy. Modeli vychislenii. Struktury dannykh* [Graphs. Computation models. Data structures]. N. Novgorod, NSU Publ., 2005. 307 p. (in Russian).