УДК 519.1:519.16

АЛГОРИТМЫ ПОИСКА ПУТИ В ГРАФАХ: АНАЛИЗ, СРАВНЕНИЕ И ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ

Худницкий А.В.[1], студент гр.353502, Клочко А.В.[2], студент гр. 353502

Белорусский государственный университет информатики и радиоэлектроники[1]
г. Минск, Республика Беларусь

Примичева З.Н.– доцент кафедры высшей математики

Аннотация. В данной статье рассматриваются алгоритмы поиска пути на графах, которые представляют собой фундаментальную область в информатике и интенсивно изучаются как в научных, так и в прикладном сферах. Проблема поиска оптимального пути в графах возникает в различных областях, включая планирование маршрутов, оптимизацию логистических процессов, разработку игр и анализ данных.

Ключевые слова. Граф, эвристика, матрица смежности, DFS, BFS, алгоритм Дейкстры, алгоритм A*, алгоритм Беллмана-Форда.

Алгоритмы поиска пути в графах играют ключевую роль во множестве приложений, начиная от популярных игр, навигационных систем и заканчивая робототехникой. Графы являются универсальной моделью для представления сложных систем, в которых вершины представляют собой объекты, а рёбра - связи между ними. Поиск оптимального пути в графе является одним из основных алгоритмических заданий, с которым сталкиваются разработчики программного обеспечения.

Цель данной работы состоит в анализе и сравнении различных алгоритмов поиска пути в графах. В рамках исследования будут рассмотрены следующие алгоритмы: алгоритм Дейкстры (1), алгоритм A* (2), алгоритм Беллмана-Форда (3), алгоритмы поиска в ширину (BFS) (4) и глубину (DFS) (5). Особое внимание будет уделено эффективности, скорости, точности и применимости каждого алгоритма в различных сценариях использования.

Анализ и сравнение различных методов поиска пути в графах позволят выявить их преимущества и ограничения, а также определить наилучшие практики для конкретных сценариев применения. Ожидается, что результаты данной работы будут полезны для разработчиков, инженеров и исследователей, стремящихся улучшить эффективность алгоритмов поиска пути в своих проектах, путем их замены или модификации.

Для каждого из пяти алгоритмов необходимо провести тестирование на разнообразных графах, которые помогут выявить сильные и слабые стороны тех или иных алгоритмов. А позже, на основании результатов, сделать вывод о том, в каких областях науки можно использовать определенный алгоритм.

Алгоритмы будут сравниваться по следующим критериям:

1. Скорость выполнения алгоритма, выраженная в количестве операций, совершенных в процессе тестирования для нахождения выбранной вершины.
2. Скорость выполнения алгоритма, выраженная в количестве операций, совершенных в процессе тестирования для отработки графа целиком.
3. Длина полученного пути, которая позволит определить, нашел ли алгоритм самый оптимизированный путь.

Будет выделено три этапа тестирования: тестирование алгоритмов на невзвешенных графах, на взвешенных графах с рёбрами неотрицательного веса, на взвешенных графах с ребрами любого веса. Каждый этап будет включать несколько графов, демонстрирующих особенности работы алгоритмов в различных условиях. Сами тесты будут представлены в виде матрицы смежности или матрицы весов.

На первом этапе будет проведено тестирование рассматриваемых алгоритмов на невзвешенных графах. Хотя далеко не все из алгоритмов рассчитаны на работу с невзвешенными графами, попробуем выяснить, насколько они эффективны в данных условиях.

Сами алгоритмы были написаны с помощью языка программирования C++. Главная часть их кода с кратким объяснением алгоритма продемонстрирована ниже на рисунках 1-5.

```
std::priority_queue<std::pair<int, int>, std::vector<std::pair<int, int>>, std::greater<std::pair<int, int>>> q; //создание приоритетной очереди q
    q.push({ 0, begin }); //добавление начальной вершины
    while(!q.empty()){ //запуск цикла, выполнение которого будет происходить до того момента, как очередь не станет пустой
        std::pair<int, int> current;
        current = q.top(); //извлечение вершины с наименьшим расстоянием от заданной
        q.pop();
        min = current.first; //сохранение расстояние в переменной min
        minindex = current.second; //сохранение индекса в переменной minindex
        for (std::pair<int, int> e : graphEdges[minindex]) { //проверка возможности улучшить расстояние до смежной вершины
            if (min + e.second < distDeikstra[e.first]) { //если сумма расстояния до текущей вершины и веса ребра меньше, чем расстояние до смежной вершины
                distDeikstra[e.first] = min + e.second; //то обновление расстояния
                q.push({ min + e.second , e.first }); // то добавление вершины в приоритетную область
            }
        }
```

Рисунок 1 - Алгоритм Дейкстры

```
std::priority_queue<std::pair<int, int>, std::vector<std::pair<int, int>>, std::greater<std::pair<int, int>>> q; //создание приоритетной очереди q
    q.push({ 0 + heuristic(begin, goal), begin}); //добавление вершины с оценочным(путь от начальной до текущей + эвристическая оценка для тех же вершин) весом в очередь
    while (!q.empty()) { //запуск цикла, который будет выполняться до того момента, как очередь не окажется пустой
        std::pair<int, int> current;
        current = q.top();//извлечение вершины с наименьшей суммарной стоимости из приоритетной очереди
        q.pop();
        min = current.first; //сохранение расстояние в переменной min
        minindex = current.second;//сохранение индекса в переменной minindex
        for (std::pair<int, int> e : graphEdges[minindex]) { //цикл, проверяющий возможность улучшить расстояния дял каждой смежной вершины
            if (distA[minindex] + e.second < distA[e.first]) {//если сумма расстояния до текущей вершины и веса ребра меньше, чем расстояние до смежной вершины
                distA[e.first] = distA[minindex] + e.second;//то обновление расстояния до смежной вершины
                q.push({ distA[e.first] + heuristic(e.first, goal) , e.first });//то добавление в очередь суммы расстояний до смежной вершины и эвристической оценки
            }
        }
```

Рисунок 2 - Алгоритм A*

```
for (int i = 0; i < numOfVerticles - 1; ++i) {//внешний цикл, идущий по всем вершинам
    for (int j = 0; j < numOfEdges; ++j) {//внутренний цикл, идущий по всем ребрам
        if (distFordBellman[e[j].a] < 10000) {//проверка
            if (distFordBellman[e[j].b] > distFordBellman[e[j].a] + e[j].cost) {
                //если расстояние до врешины e[j].b больше расстояние до e[j].a
                distFordBellman[e[j].b] = distFordBellman[e[j].a] + e[j].cost;
                //то означает, что был найден более короткий путь
            }
        }
    }
}
```

Рисунок 3 - Алгоритм Беллмана-Форда

```
while (!queueBfs.empty()) {//цикл, выполняющийся, пока очередь не станет пустой
        int cur = queueBfs.front();//занесение в переменную cur вершину из передней части очереди
        queueBfs.pop();//после извлечения она удаляется
        for (std::pair<int, int> neighbor : graphEdges[cur]) {//цикл для обхода смежных вершин
            if (!usedBfs[neighbor.first]) {//проверка на то, что вершина не была уже посещена
                distBfs[neighbor.first] = distBfs[cur] + 1;//увеличение расстояние на 1
                queueBfs.push(neighbor.first);//добавление вершины в конец очереди
                usedBfs[neighbor.first] = true;//помечка того, что вершина уже была посещена
            }
        }
    }
```

Рисунок 4 - Алгоритм поиска в ширину

```
for (std::pair<int, int> u : graphEdges[v]) {//цикл, выполняющийся для каждого
                                             //угла в списке смежных узлов
        if (!usedDfs[u.first]) {//проверка на повещение угла ранее
            distDfs[u.first] = distDfs[v] + 1;//увеличение расстояния на 1
            dfs(u.first);//рекусривный вызов для первого узла
        }
    }
```

Рисунок 5 - Алгоритм поиска в глубину

Тест №1 представляет собой стандартный ориентированный граф из 20 вершин, который показывает базовую работу алгоритмов. Матрица смежности для теста №1 показана ниже на рисунке 6.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0
6	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
11	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
12	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
19	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Рисунок 6 - Матрица смежности для теста №1

Таблица 1 - Результаты теста №1

	(1)	(2)	(3)	(4)	(5)
Кол-во шагов от 1 до 20 элемента	35	19	35	36	16
Общее кол-во шагов обхода графа	38	47	722	38	38
Длина пути от 1 до 20 элемента	5	5	5	5	7

В данном тесте наблюдается преимущество в скорости алгоритмов A* и поиска в глубину при нахождении искомого элемента, однако алгоритм поиска в глубину при этом находит не самый кратчайший путь из возможных. Общее количество шагов обходов графа говорит о том, сколько каждый из алгоритмов делает итераций при обходе всего графа и соответственно тратит времени на свою работу. Алгоритм Беллмана-Форда демонстрирует значительные затраты на обход графа, что связано с тем, что в данном случае алгоритм 19 раз рассматривает абсолютно каждое ребро. Это свидетельствует о его неэффективности при использовании в невзвешенных графах. Алгоритм A* при этом затрачивает большее количество итераций в процессе обхода графа, но находит необходимый элемент одним из первых.

Также стоит отметить, что пути нахождения нужного элемента алгоритмами отличаются, несмотря на то, что длина пути для большинства алгоритмов одинакова. Это происходит из-за наличия эвристической оценки у алгоритма A*, а также специфики подбора рассматриваемого элемента у алгоритма Дейкстры. Проделанный алгоритмами путь до одной из вершин показан на рисунке 7.

```
Distances from 1 to 18:
DFS: 5
BFS: 4
Deikstra: 4
A*: 4
Bellman-Ford: 4

Path for the DFS:
1 6 5 8 9 18
Path for the BFS:
1 6 7 10 18
Path for the Deikstra:
1 5 8 9 18
Path for the A*:
1 6 7 12 18
Path for the Bellman-Ford:
1 5 8 9 18
```

Рисунок 7 - Путь алгоритмов от 1 до 18 вершины

Тест №2 представляет собой неориентированный граф, состоящий из 20 вершин. Он используется, чтобы посмотреть работу алгоритмов при других входных данных. Матрица весов для теста №2 показана ниже на рисунке 8.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	1	1	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0
2	1	0	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	1	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
4	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
6	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
9	1	1	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
10	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	1	0	0	0
11	1	0	1	0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	0	0
12	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	1	0	0
13	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1
15	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	0	0
17	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0	1	0	0
18	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
20	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0

Рисунок 8 - Матрица смежности для теста №2

Таблица 2 - Результаты теста №2

	(1)	(2)	(3)	(4)	(5)
Кол-во шагов от 1 до 20 элемента	77	43	73	80	59
Общее кол-во шагов обхода графа	84	91	1596	84	84
Длина пути от 1 до 20 элемента	4	4	4	4	15

Результаты теста №2 схожи с результатами теста №1: алгоритм A* сработал быстрее всех, а общее количество шагов в алгоритме Беллмана-Форда также огромно по сравнению с другими алгоритмами. Однако длина пути для DFS превосходит оптимальный путь более чем в 3 раза. Это связано с тем, что данный алгоритм выбирает путь случайно, продолжая его до тех пор, пока не дойдет до нужной вершины или не зайдет в тупик. Из этого следует, что поиск в глубину при неблагоприятных условиях может найти далеко не самый оптимальный путь, в отличие от других алгоритмов поиска пути. Стоит заметить, что при этом DFS по скорости лучший после алгоритма A*. Именно этот алгоритм показывает на данный момент наилучшие результаты по скорости и оптимальности.

Теперь проверим алгоритмы с помощью взвешенных графов, имеющих неотрицательные веса ребер. Проверим, какие результаты будут выдавать алгоритмы на данном этапе.

В тесте №3 предложен неориентированный взвешенный граф из 20 вершин. Для него, а также для всех последующих тестов, будет использоваться матрица весов для демонстрации значений веса каждого из ребер графа. Матрица весов для теста №3 показана ниже.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	2	0	0	0	0	0	0	0	5	0	0	0	3	0	0	0	0	0	0
2	2	0	1	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	0	2	2	0	0	0	3	0	6	0	0	0	0	0	0	0	0	0
4	0	0	2	0	4	0	0	0	4	0	3	0	0	0	0	0	0	0	0	0
5	0	2	2	4	0	0	0	0	4	0	1	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	4	0	6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	3	2	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	2
9	0	0	3	4	4	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
10	5	0	0	0	0	0	0	0	0	0	0	0	0	0	3	4	0	0	1	0
11	0	0	6	3	1	4	0	0	2	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	3	0	0	0	0	0	0	2	0	0	2	3	0	0
13	0	0	0	0	0	6	0	0	0	0	0	0	0	0	2	3	0	0	0	0
14	3	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	1	0	0	0
15	0	0	0	0	0	0	0	0	0	3	0	0	2	0	0	0	0	0	3	0
16	0	0	0	0	0	0	0	0	0	4	0	0	3	0	0	0	0	0	0	3
17	0	0	0	0	0	0	3	2	0	0	0	2	0	1	0	0	0	3	0	0
18	0	0	0	0	0	0	2	0	0	0	0	3	0	0	0	0	3	0	0	0
19	0	0	0	0	0	0	0	0	0	1	0	0	0	0	3	0	0	0	0	0
20	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	3	0	0	0	0

Рисунок 9 - Матрица весов теста №3

Таблица 3 - Результаты теста №3

	(1)	(2)	(3)	(4)	(5)
Кол-во шагов от 1 до 20 элемента	43	13	109	40	33
Общее кол-во шагов обхода графа	68	81	1292	68	68
Длина пути от 1 до 20 элемента	8	8	8	12	37

В случае же придания ребрам положительного веса количество шагов для алгоритма Беллмана-Форда сильно возрастает. Связано это с тем, что у самого алгоритма высокая сложность поиска пути и огромное количество итераций. Алгоритм A* в очередной раз показывает лучший результат. Эвристическая составляющая значительно повышает эффективность алгоритма в большом

количестве случаев. В качестве эвристики было выбрано манхэттенское расстояние, которое работает с заданными для вершин координатами. Пример задания вершин в программе показан на рисунке 10.

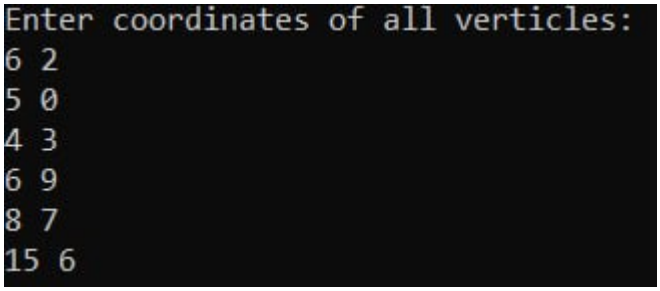

Рисунок 10 - Задание координат вершин графа

Хоть количество шагов от 1 до 20 элемента для алгоритма поиска в глубину и в длину оказалось меньшим, чем у алгоритма Дейкстры, сам путь оказался далеко не оптимизированным, что связано с принципом работы каждого из алгоритмов.

Особенностью теста №4 является то, что искомый путь может быть найден несколькими способами, большинство из которых являются затратными и длинными.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0
2	1	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
3	1	2	0	4	0	0	0	0	0	0	0	0	0	0	2	2	0	0	0	0
4	0	0	4	0	0	0	2	0	0	0	0	0	0	3	0	0	5	4	0	0
5	0	0	0	0	0	0	0	4	6	0	0	0	1	2	0	3	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	3	1
7	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	4	6	0
8	0	0	0	0	4	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	6	0	0	1	0	1	0	0	0	2	0	0	0	0	0	0
10	0	0	0	0	0	0	0	1	1	0	0	0	5	3	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	5	2	2	6	6	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	9
13	0	0	0	0	1	2	0	0	0	5	0	0	0	0	0	0	0	0	3	0
14	0	0	0	3	2	0	0	0	2	3	5	0	0	0	0	0	0	0	0	0
15	0	0	2	0	0	0	0	0	0	0	2	0	0	0	0	4	0	0	0	0
16	0	0	2	0	3	0	0	0	0	0	2	0	0	0	4	0	0	0	0	0
17	3	2	0	5	0	0	0	0	0	0	6	4	0	0	0	0	0	0	0	0
18	0	0	0	4	0	0	4	0	0	0	6	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	3	6	0	0	0	0	0	3	0	0	0	0	0	0	0
20	0	0	0	0	0	1	0	0	0	0	0	9	0	0	0	0	0	0	0	0

Рисунок 11 - Матрица весов теста №4

Таблица 4 - Результаты теста №4

	(1)	(2)	(3)	(4)	(5)
Кол-во шагов от 1 до 20 элемента	55	63	113	35	40
Общее кол-во шагов обхода графа	74	97	1406	74	74
Длина пути от 1 до 20 элемента	10	10	10	16	37

Тестирование показало отличный от предыдущих тестов результат. Алгоритм A* совершил большее количество шагов, чем алгоритм Дейкстры. Связано это с определенным расположением вершин графа и весов его ребер, которые позволяют “обхитрить” эвристику алгоритма A*. Поиск в ширину нашел путь от 1 до 20 вершины раньше всех, так как искомая вершина расположена близко к заданной. Алгоритм Беллмана-Форда по скорости работы отстает от других алгоритмов минимум в 3 раза, что связано с большим количеством ребер.

В заключительном этапе тестирования будут проверены алгоритмы графов с ребрами произвольного веса, в том числе и отрицательного.

Тесты №5 и №6 в отличие от предыдущих тестов имеют отрицательные ребра, что может повлиять на корректную работу алгоритмов. Матрица весов для тестов №5 и №6 представлена на рисунке 12 ниже.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	0	0	0	0	0	0	0	0	3	0	1	0	0	0	0	0	0	0	0
2	-2	0	0	0	0	0	0	0	0	0	0	6	-4	0	0	0	8	0	0	0
3	0	0	0	0	0	2	0	0	3	0	0	0	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	-3	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
6	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	2	0	0
7	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	1
9	0	0	3	0	0	0	0	0	0	0	1	0	0	0	-1	0	0	0	0	0
10	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	3	0	0	1	0	0	0	0	0	0	3	0	0	0	0
12	0	6	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	2	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-3	0	1	0
14	0	0	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
16	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	-2	0
17	0	0	0	1	0	0	0	0	4	2	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	-4	0	0	-3	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	1	0	0	0	1	0	0	0	2	0	0	0	0	0
20	0	0	0	0	3	0	0	1	0	0	0	9	0	0	2	0	0	0	0	0

Рисунок 12 - Матрица весов для тестов №5 и №6

Таблица №5 - Результаты теста №5

	(1)	(2)	(3)	(4)	(5)
Кол-во шагов от 1 до 20 элемента	36	18	131	36	12
Общее кол-во шагов обхода графа	44	44	836	44	44
Длина пути от 1 до 20 элемента	15	17	6	15	25

По результатам теста можно сделать вывод, что алгоритмы Дейкстры и A* начинают считать далеко не самый оптимальный ответ. Соответственно, для работы с отрицательными рёбрами эти алгоритмы в своём классическом представлении плохо приспособлены.

Слегка модифицировав алгоритмы Дейкстры и A* так, что они могут заново просматривать уже обработанную вершину, были получены следующие ответы на предыдущем графе.

Таблица №6 - Результаты теста №6

	(1)	(2)	(3)	(4)	(5)
Кол-во шагов от 1 до 20 элемента	52	94	131	36	12
Общее кол-во шагов обхода графа	55	117	836	44	44
Длина пути от 1 до 20 элемента	6	6	6	15	25

С данной модификацией алгоритмы Дейкстры и A* теперь показывают ответ с наименьшей возможной суммой, однако при таком же количестве операций, требуемых для полного обхода графа. увеличилось.

Также стоит отметить, что в данном тесте алгоритм A* снова показал результат хуже, чем у алгоритма Дейкстры, что говорит о трудностях подбора правильной и эффективной эвристики.

Тест №7 имеет отрицательные циклы, что может негативно повлиять на работу алгоритмов. Матрица весов для теста №7 показана на рисунке 13.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	0	0	0	0	2	0	0	0	3	0	0	0	1	3	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	5	0	6	0	0	4	0	0	0	0
3	0	0	0	0	0	0	0	0	2	4	0	0	0	0	0	0	0	2	0	0
4	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	-6	0	0	4	0
5	0	0	0	0	0	0	0	0	-3	0	0	0	0	0	0	5	0	0	0	0
6	2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
8	0	1	0	0	0	0	0	0	0	0	0	-4	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	8	0
10	3	0	4	0	0	0	0	0	0	0	-2	0	0	-4	0	0	0	0	0	0
11	0	5	0	0	0	0	0	0	-1	0	0	0	0	0	0	-3	0	0	0	0
12	0	0	0	0	0	0	8	-4	0	0	0	0	4	0	0	0	0	0	0	10
13	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7
14	1	0	0	0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	1	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0	0	0
16	0	4	0	-6	5	0	0	0	0	0	-3	0	0	0	0	0	5	0	0	0
17	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	2	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	-2	0	0	0	3	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0

Рисунок 13 - Матрица весов теста №7

Таблица №7 - Результаты теста №7

	(1)	(2)	(3)	(4)	(5)
Кол-во шагов от 1 до 20 элемента	30	25	967	27	25
Общее кол-во шагов обхода графа	51	51	969	51	51
Длина пути от 1 до 20 элемента	1	1	-196	25	27

В данном тесте использовались классические алгоритмы Дейкстры и A*, так как модифицированные их варианты неспособны корректно работать из-за наличия циклов с отрицательными рёбрами. Алгоритм Беллмана-Форда в свою очередь находит наименьшее расстояние до искомой вершины, однако делает он это при помощи вечного обхода по отрицательным циклам. Результат находится лишь благодаря конечному числу итераций алгоритма, при увеличении которых ответ будет соответственно уменьшаться.

После проведенных тестов может показаться, что алгоритм Беллмана-Форда проигрывает по сравнению с остальными алгоритмами, но это не так. Сильная сторона алгоритма заключается в способности оптимально работать с большим количеством вершин, что продемонстрировано на рисунке 14[1].

Рисунок 14 - График зависимости скорости работы алгоритма Беллмана-Форда и алгоритма Дейкстры от количества вершин

Несмотря на то, что алгоритм Дейкстры является одним из лучших алгоритмов для поиска пути, он проигрывает алгоритму Беллмана-Форда для случаев с большим количеством вершин графа.

На основе проведенного тестирования алгоритмов можно судить о сильных и слабых сторонах каждого алгоритма и сферах их применения.

Алгоритм Дейкстры - простой и эффективный алгоритм поиска пути во взвешенных графах, который легко можно модифицировать под конкретную задачу, а единственным его недостатком является возможность корректной работы только с неотрицательными весами ребер. Неудивительно, что он популярен во многих областях. Например, именно он используется для одноадресной маршрутизации, а именно для нахождения наименьшего расстояния между любыми узлами сети. Алгоритм также используется для поиска кратчайшего пути между пользователями в социальной сети, однако с увеличением размеров графа приоритет отдается расширенным алгоритмам. И, конечно же, данный алгоритм используется для GPS-навигации, а именно для маршрутов наименьшей длины между указанными точками.

Алгоритм можно использовать и для новейших открытий человечества: обмена информацией в интернете вещей и в умных городах. Например, алгоритм может применяться для оптимизации транспортных маршрутов, управления светофорами, тем самым повышая безопасность дорог и уменьшая количество потенциальных аварий.

Алгоритм A* очень схож с алгоритмом Дейкстры, однако имеет особенность: эвристическую функцию, которая позволяет оценить оставшееся расстояние до цели. Из-за того, что саму эвристическую функцию можно задавать исходя из условий и целей задачи, алгоритм имеет очень большую мобильность и широкий спектр применения. Например, он используется в игровой индустрии для определения передвижений игрового персонажа. Подобное используется и в робототехнике: алгоритм A* определяет маршрут роботов на фабриках, складах и пространстве в целом. Большую популярность алгоритм получил со сверхбыстрым развитием искусственного интеллекта за ближайшие 5 лет, так как его поведение и действия задаются с помощью алгоритма A*.

Алгоритм Беллмана-Форда, в отличие от предыдущих алгоритмов, может работать и с отрицательными весами, что обеспечивает ему широкое использование среди программистов. Однако алгоритм имеет высокую вычислительную сложность, что делает его использование при работе с небольшими графами нецелесообразным. В случае же больших графов данный алгоритм показывает себя очень достойно. Он часто используется для маршрутизации, транспортной логистики и робототехники. Интересно, что алгоритм также используется в финансовой аналитике: он позволяет моделировать финансовые потоки, а затем на их основе определять оптимальную финансовую стратегию.

BFS (поиск в ширину) - алгоритм, отличительной особенностью которого является то, что он в порядке очереди обходит каждую вершину графа, что делает его наличие обязательным для решения специфичных задач. Например, данный алгоритм используется для моделирования распространения стихийных бедствий: пожара, извержения вулкана, наводнения. Поиск в ширину используется для решения различных головоломок и задач на комбинаторику.

BFS часто используется для нахождения кратчайшего пути из лабиринта. Это применимо в играх, суть которых создания лабиринтов разной сложности и их прохождение. Также их можно использовать для генерации случайных данных, которые далее будет собирать BFS и для проверки навыков искусственного интеллекта.

DFS (поиск в глубину) - алгоритм, особенностью которого является то, что он обходит каждую из доступных вершин на максимальную глубину. Он применим, также как и BFS, в машинном обучении. Алгоритм также используется для определения максимального потока транспорта, анализа древовидных структур. DFS также может использоваться для анализа исходного кода программы. Например, он применим для поиска определенных переменных или вызовов функций.

Список использованных источников:

1. Сравнительный анализ алгоритмов поиска пути в помещении [электронный ресурс]. – Режим доступа : https://cyberleninka.ru/article/n/sravnitelnyy-analiz-algoritmov-poiska-puti-v-pomeschenii/viewer

2. Алгоритм Форда-Беллмана [электронный ресурс]. – Режим доступа : https://e-maxx.ru/algo/ford_bellman

3. Введение в алгоритм A [электронный ресурс]. – Режим доступа : https://habr.com/ru/articles/331192/*

UDC 519.1:519.16

PATHFINDING ALGORITHMS IN GRAPHS: ANALYSIS, COMPARISON AND PRACTICAL USAGE

Hudnitskii A.V. [1], *Klochko A.V.* [2]

Belarusian State University of Informatics and Radioelectronics[1], *Minsk, Republic of Belarus*

Primicheva Z.N. – associate professor of the department of higher mathematics

Annotation. This article describes graph pathfinding algorithms, which represent a fundamental field in computer science and are intensively studied in both academic and applied contexts. The problem of finding the optimal path in graphs arises in various fields, including route planning, optimization of logistics processes, game development and data analysis.

Keywords. Graf, heuristics, adjacency matrix, DFS, BFS, Deikstra algorithm, A* algorithm, Bellman-Ford algorithm.