СБОРНИК **НАУЧНЫХ ТРУДОВ НГТУ. – 2017. – № 2 (88).** – 33–40

УДК 519.6

АЛГОРИТМ ПОИСКА ПУТИ ИЗ ПУНКТА А В ПУНКТ Б*

Е.П. МИКОВ[1], В.А. БОНДАРЬ[2]

[1] *630073, РФ, г. Новосибирск, пр. Карла Маркса, 20, Новосибирский государственный технический университет, студент кафедры автоматики. E-mail: bondar.vale@mail.ru*
[2] *630073, РФ, г. Новосибирск, пр. Карла Маркса, 20, Новосибирский государственный технический университет, студент кафедры автоматики. E-mail: Mikov.e.p@gmail.com*

Поиск пути из точки А в точку Б – одна из самых распространенных задач при разработке игр. Для решения этой задачи есть множество алгоритмов, но программа, разработанная нами, является уникальным вариантом реализации метода поиска пути. Многие программисты и разработчики игр не владеют информацией на предмет того, какой алгоритм поиска пути из двух отдаленных точек наиболее эффективен. Чтобы реализовать задуманный проект, возникает необходимость ознакомления с множеством алгоритмов, из которых далеко не каждый удобен в пользовании. В ходе написания нашей программы были испробованы многие методы и в итоге определены самые совершенные и, как оказалось, наиболее популярные алгоритмы поиска, именуемые A* (A star). Цель работы – создание драйвера-программы для роботов, основанных на схемах Arduin, для нахождения кратчайшего пути из точки А в точку Б. Следует отметить, что у схем Arduin есть такой недостаток, как отсутствие многопоточности. В связи с этим в ходе реализации проекта появилась еще одна цель – это оптимизация алгоритма для одного потока процесса. Следовательно, нашей конечной целью можно назвать способ написания однопоточного алгоритма.

Ключевые слова: алгоритмизация, поиск пути, C#, программа, однопоточность, оптимизация, Arduin, разработка игр

DOI: 10.17212/2307-6879-2017-2-33-40

ВВЕДЕНИЕ

Многие программисты и разработчики игр не владеют информацией на предмет того, какой алгоритм поиска пути из двух отдаленных точек наиболее эффективен. Чтобы реализовать задуманный проект, возникает необходимость ознакомления с множеством алгоритмов, из которых далеко не каждый удобен в пользовании. В ходе написания программы мы испробовали многие

* Статья получена 02 мая 2017 г.

методы и в итоге определили самый совершенный и, как оказалось, наиболее популярный способ алгоритмизации поиска, именуемый A* (A star). Ему и посвящена наша работа.

1. ЦЕЛЬ ПРОДЕЛАННОЙ РАБОТЫ И РЕАЛИЗАЦИЯ МЕТОДОВ АЛГОРИТМИЗАЦИИ ПОИСКА

Цель работы – создание драйвера-программы для роботов, основанных на схемах Arduin, для нахождения кратчайшего пути из точки А в точку Б. Следует отметить, что у схем Arduin есть такой недостаток, как отсутствие многопоточности. В связи с этим в ходе реализации проекта у нас появилась еще одна цель – это оптимизация алгоритма для одного потока процесса. Следовательно, нашей конечной целью можно назвать разработку способа написания однопоточного алгоритма, который будет выполнять следующее:

1) переработку местности в массив;

2) сжатие массива до максимально возможного размера жесткости;

3) поиск кратчайшего пути из точки А в точку Б.

Ниже изложены особенности метода A* (A star) и предложенный нами способ реализации.

Как и алгоритм поиска в ширину, алгоритм A* является полным в том смысле, что он всегда находит решение, если таковое существует.

Если эвристическая функция h допустима, т. е. никогда не переоценивает действительную минимальную стоимость достижения цели, то алгоритм A* сам является допустимым (или оптимальным) также при условии, что мы не отсекаем пройденные вершины. Если же мы это делаем, то для оптимальности алгоритма требуется, чтобы $h(x)$ была еще и монотонной или преемственной эвристикой. Свойство монотонности означает, что если существуют пути A–B–C и A–C (необязательно через B), то оценка стоимости пути от A до C должна быть меньше суммы оценок путей A–B и B–C либо равна этой сумме. (Монотонность также известна как неравенство треугольника: одна сторона треугольника не может быть длиннее, чем сумма двух других сторон.) Математически для всех путей x, y (где y – потомок x) выполняется условие

$$g(x) + h(x) <= g(y) + h(y).$$

Алгоритм A* также оптимально эффективен для заданной эвристики h. Это значит, что любой другой алгоритм исследует не меньше узлов, чем A* (за исключением случаев, когда существует несколько частных решений с одинаковой эвристикой, точно соответствующей стоимости оптимального пути).

В то время как алгоритм A* оптимален для «случайно» заданных графов, нет гарантии, что он сделает свою работу лучше, чем более простые, но и более информированные относительно проблемной области алгоритмы. Например, в неком лабиринте может потребоваться сначала идти по направлению от выхода и только потом повернуть назад. В этом случае обследование вначале тех вершин, которые расположены ближе к выходу (по прямой дистанции), будет потерей времени.

Особые случаи

Поиск в глубину и поиск в ширину являются двумя частными случаями алгоритма A*. Для поиска в глубину возьмем глобальную переменную–счетчик C, инициализировав ее неким большим значением. Всякий раз при раскрытии вершины будем присваивать смежным вершинам значение счетчика, уменьшая его на единицу после каждого присваивания. Таким образом, чем раньше будет открыта вершина, тем большее значение $h(x)$ она получит, а значит, будет просмотрена в последнюю очередь. Если положить $h(x) = 0$ для всех вершин, то мы получим еще один специальный случай – алгоритм Дейкстры.

Тонкости реализации

Существует несколько особенностей реализации и приемов, которые могут значительно повлиять на эффективность алгоритма. Первое, на что следует обратить внимание, – это то, как очередь с приоритетом обрабатывает связи между вершинами. Если вершины добавляются в нее так, что очередь работает по принципу LIFO, то в случае вершин с одинаковой оценкой алгоритм A* «пойдет» в глубину. Если же при добавлении вершин реализуется принцип FIFO, то для вершин с одинаковой оценкой алгоритм, напротив, будет реализовывать поиск в ширину. В некоторых случаях это обстоятельство может оказывать существенное влияние на производительность.

В случае, если по окончании работы требуется сохранение, то вместе с каждой вершиной обычно хранят ссылку на родительский узел. Эти ссылки позволяют реконструировать оптимальный маршрут. Если так, тогда важно, чтобы одна и та же вершина не встречалась в очереди дважды (имея при этом свой маршрут и свою оценку стоимости). Обычно для решения этой проблемы при добавлении вершины проверяют, нет ли записи о ней в очереди. Если она есть, то запись обновляют так, чтобы она соответствовала минимальной стоимости. Для поиска вершины в сортирующем дереве многие стандартные алгоритмы требуют времени $O(n)$. Если усовершенствовать дерево с помощью хеш-таблицы, то можно уменьшить это время.

Почему алгоритм A* допустим и оптимален

Алгоритм A* и допустим, и обходит при этом минимальное количество вершин, благодаря тому что работает с «оптимистичной» оценкой пути через вершину. Оптимистичной в том смысле, что если он пойдет через эту вершину, то у него «есть шанс», что реальная стоимость результата будет равна этой оценке, но никак не меньше. Поскольку A* является информированным алгоритмом, такое равенство может быть вполне возможным.

Когда алгоритм A* завершает поиск, то, согласно определению, он находит путь, истинная стоимость которого меньше, чем оценка стоимости любого пути через любой открытый узел. Но поскольку эти оценки являются оптимистичными, соответствующие узлы можно без сомнений отбросить. Иначе говоря, A* никогда не упустит возможности минимизировать длину пути и потому является допустимым.

Предположим теперь, что некий алгоритм Б вернул в качестве результата путь, длина которого больше оценки стоимости пути через некоторую вершину. На основе эвристической информации для алгоритма Б нельзя исключить возможность, что этот путь имел и меньшую реальную длину, чем результат. Соответственно, пока алгоритм Б просмотрел меньше вершин, чем A*, он не будет допустимым. Итак, A* проходит наименьшее количество вершин графа среди допустимых алгоритмов, использующих такую же точную (или менее точную) эвристику.

Оценка сложности

Временная сложность алгоритма A* зависит от эвристики. В худшем случае число вершин, исследуемых алгоритмом, растет экспоненциально по сравнению с длиной оптимального пути, но сложность становится полиномиальной, когда эвристика удовлетворяет следующему условию:

$$|h(x) - h^*(x)| <= O(\log h^*(x)),$$

где h^* – оптимальная эвристика, т. е. точная оценка расстояния из вершины x к цели. Другими словами, ошибка $h(x)$ не должна расти быстрее, чем логарифм от оптимальной эвристики.

Но еще большую проблему, чем временная сложность, представляют собой потребляемые алгоритмом ресурсы памяти. В худшем случае ему приходится помнить экспоненциальное количество узлов. Для борьбы с этим было предложено несколько вариаций алгоритма, таких как алгоритм A* с итеративным углублением (iterativedeeping A*, IDA*), A* с ограничением памяти (memory-bounded A*, MA*), упрощённый MA* (simplified MA*, SMA*) и рекурсивный поиск по первому наилучшему совпадению (recursivebest-firstsearch, RBFS).

2. АЛГОРИТМ

1. Создается два списка вершин – ожидающие рассмотрения и уже рассмотренные. В ожидающие добавляется точка старта, список рассмотренных пока пуст.

2. Для каждой точки рассчитывается $F = G + H$, где G – расстояние от старта до точки, H – примерное расстояние от точки до цели. Также каждая точка хранит ссылку на точку, из которой в нее пришли.

3. Из списка точек на рассмотрение выбирается точка с наименьшим F. Обозначим ее X.

4. Если X – цель, то мы нашли маршрут.

5. Переносим X из списка ожидающих рассмотрения в список уже рассмотренных.

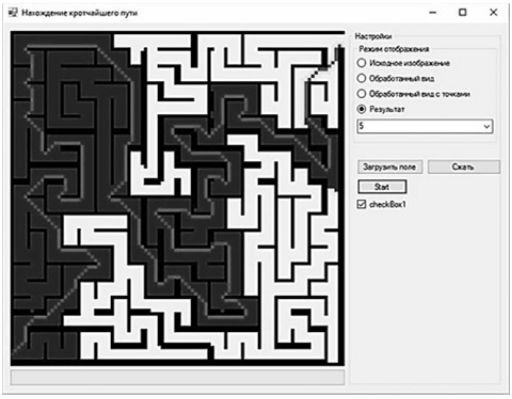
6. Для каждой из точек, соседних для X (обозначим эту точку Y), делаем следующее:

– если Y уже находится в рассмотренных – пропускаем ее;

– если Y еще нет в списке на ожидание – добавляем ее туда, запомнив ссылку на X и рассчитав $Y.G$ (это $X.G$ + расстояние от X до Y) и $Y.H$;

– если же Y в списке на рассмотрение – проверяем, если $X.G$ + расстояние от X до $Y < Y.G$, значит мы пришли в точку Y более коротким путем, заменяем $Y.G$ на $X.G$ + расстояние от X до Y, а точку, из которой пришли в Y, на X;

– если список точек на рассмотрение пуст, а до цели мы так и не дошли – значит маршрут не существует (рисунок).



Результат работы алгоритма

ЗАКЛЮЧЕНИЕ

В данной статье мы постарались наглядно осветить основные принципы функционирования взвешенных графов и методов их создания. Программа позволяет находить кратчайшее расстояния из точки А в точку Б. Дальнейшее усложнение этой программы позволит анализировать более сложные клеточные пространства и графы, а также находить кратчайший маршрут между ними.

БЛАГОДАРНОСТИ

Авторы выражают искреннюю благодарность профессору кафедры автоматики А.А. Воеводе за помощь при написании работы, а также за полезное обсуждение полученных результатов.

СПИСОК ЛИТЕРАТУРЫ

1. *Рихтер Д.* CLR via C#: программирование на платформе Microsoft .NET Framework 4.5 на языке C#. – 4-е изд. – Спб.: Питер, 2017. – 896 с.

2. *Дейтел П., Дейтел Х.* Как программировать на Visual C# 2012: включая работу в Windows 7 и Windows 8. – 5-е изд. – СПб.: Питер, 2014. – 585 с.

3. *Мак-Дональд М.* WPF: Windows Presentation Foundation в .NET 4.5 с примерами на C# 5.0: для профессионалов. – 4-е изд. – М.: Вильямс, 2012. – 1024 с.

4. *Павловская Т.А.* C#: программирование на языке высокого уровня. – СПб.: Питер, 2009. – 432 с.

5. *Евстигнеев В.А.* Итеративные алгоритмы глобального анализа графов. Пути и покрытия // Применение теории графов в программировании / под ред. А.П. Ершова. – М.: Наука, 1985. – Гл. 3. – С. 138–150.

6. *Таланов В.А.* Нахождения кратчайших путей в графе // Алексеев В.Е., Таланов В.А. Графы. Модели вычислений. Структуры данных. – Н. Новгород: Изд-во Нижегор. гос. ун-та, 2005. – Гл. 3.4. – С. 236–237. – ISBN 5–85747–810–8.

7. *Галкина В.А.* Построение кратчайших путей в ориентированном графе // Галкина В.А. Дискретная математика. Комбинаторная оптимизация на графах. – М.: Гелиос АРВ, 2003. – Гл. 4. – С. 75–94. – ISBN 5–85438–069–2.

8. *Берж К.* Задача о кратчайшем пути // Берж К. Теория графов и ее применения / под ред. И.А. Вайнштейна. – М.: Издательство иностранной литературы, 1962. – Гл. 7. – С. 75–81.

Миков Егор Петрович, студент факультета автоматики и вычислительной техники Новосибирского государственного университета. E-mail: Mikov.e.p@gmail.com

Бондарь Валерий Александрович, студент факультета автоматики и вычислительной техники Новосибирского государственного университета. E-mail: bondar.vale@mail.ru

The algorithm for finding the path from point A to point B*

E.P. Mikov[1], V.A. Bondar[2]

[1] *Novosibirsk State Technical University, 20 Karl Marks Avenue, Novosibirsk, 630073, Russian Federation, students of the Department of Automation. E-mail: bondar.vale@mail.ru*
[2] *Novosibirsk State Technical University, 20 Karl Marks Avenue, Novosibirsk, 630073, Russian Federation, students of the Department of Automation. E-mail:Mikov.e.p@gmail.com*

Finding a path from point A to point B is one of the most common tasks in developing games. To solve this problem, there are many algorithms, but the program developed byu sis a uniquevariant of the implementation of the path-finding method. Many programmer sand game developers do not know the in for mation about which algorithm for finding a path from two remote points is most effective. To realize the conceived project, they need to familiarize them selves with a lot of algorithms, of which not every one is comfortable touse. During the writing four program, we tried many methods, and eventually determined the most perfect and, asitturned out, the most popular way of algorithmizing the search, called A * (A star). The purpose of our joint work was to create a driver program for robots based on Arduins chemes to find the shor test path from point A to point B. It should be noted that Arduins chemes have such a disadvantage a slack of multithreading. In this regard, we have another goal in the course of implement in go ur project: optimization of the algorithm for a single process flow. There fore, our final go alis the way to write a single-threaded algorithm.

Keywords: algorithmization, path search, C#, program, singlethread, optimization, Arduin, game development

DOI: 10.17212/2307-6879-2017-2-33-40

REFERENCE

1. Richter J. *CLR via C#: programmirovanie na platforme Microsoft .NET Framework 4.5 na yazyke C#* [CLR via C#. Programming on the Microsoft .NET Framework 4.5 in C#. 4th ed. St. Petersburg, Piter Publ., 2017. 896 p. (In Russian).

2. Deitel P., Deitel H. *Kak programmirovat' na Visual C# 2012: vklyuchaya rabotu v Windows 7 i Windows 8* [Visual C# 2012. How to program]. 5th ed. St. Petersburg, Piter Publ., 2014. 585 p. (In Russian).

* *Received 02 May 2017.*

3. MacDonald M. *WPF: Windows Presentation Foundation v .NET 4.5 s primerami na C# 5.0: dlya professionalov* [WPF: Windows Presentation Foundation in .NET 4.5 with examplesin C# 5.0 for professionals]. 4th ed. Moscow, Vil'yams Publ., 2012. 1024 p. (In Russian).

4. Pavlovskaya T.A. *C#: programmirovanie na yazyke vysokogo urovnya* [C#: programming in a high-level language]. St. Petersburg, Piter Publ., 2009. 432 p.

5. Evstigneev V.A. Iterativnye algoritmy global'nogo analiza grafov. Puti i pokrytiya [Iteractive algorithms for global analysis of graphs. Ways and coverings]. *Primenenie teorii grafov v programmirovanii* [Application of graph theory in programming]. Ed. A.P. Ershov. Moscow, Nauka Publ., 1985, ch. 3, pp. 138–150.

6. Talanov V.A. Nakhozhdeniya kratchaishikh putei v grafe [Finding the shortest paths in a graph]. Alekseev V.E., Talanov V.A. *Grafy. Modeli vychislenii. Struktury dannykh* [Graphs. Models of calculations. Datastructures]. Nizhni Novgorod, National Research Lobachevsky State University of Nizhni Novgorod Publ., 2005, ch. 3.4, pp. 236–237. ISBN 5-85747-810-8.

7. Galkina V.A. Postroenie kratchaishikh putei v orientirovannom grafe [Construction of shortest paths in an oriented graph]. Galkina V.A. *Diskretnaya matematika. Kombinatornaya optimizatsiya na grafakh* [Discrete mathematics. Combinatorial optimization on graphs]. Moscow, Gelios ARV Publ., 2003,ch. 4, pp. 75–94. ISBN 5-85438-069-2.

8. Berge K. Zadacha o kratchaishem puti [The problem of the shortest path]. Berge K. *Teoriya grafov i ee primeneniya* [Theory of graphs and its applications]. Ed. by I.A. Vainshtein. Moscow, Izdatel'stvo inostrannoi literatury Publ., 1962, ch. 7, pp. 75–81. (In Russian).