УДК 519.178
П.С. Шолом, Н.В. Здолбіцька
Луцький національний технічний університет

АНАЛІЗ АЛГОРИТМІВ ОБХОДУ ГРАФА ДЛЯ ЗАДАЧІ ТРАСУВАННЯ МАРШРУТУ

Проаналізовано алгоритми обходу графа, зроблено оцінку переваг і недоліків використання розглянутих алгоритмів для задачі трасування маршруту.

Ключові слова: граф, трасування маршруту, найкоротший шлях, обхід перешкод.

Задача трасування маршруту використовується в багатьох галузях науки і техніки, зокрема у комп'ютерних мережах, задачах пошуку виходу із лабіринту, при написанні тактичних та стратегічних ігор, у хвильовому алгоритмі для трасування друкованих плат. Алгоритми обходу графа використовують в алгоритмі Форда-Фалкерсона, який вирішує задачу знаходження максимального потоку в транспортній мережі, у алгоритмах пошуку одно- і двозв'язних компонент, топологічному сортуванні. Тому знання алгоритмів обходу графа, переваги та недоліки у їх застосуванні в різноманітних задачах є досить актуальним для програмістів у час стрімкого розвитку комп'ютерної науки і техніки.

З точки зору абстрактного уявлення граф G – це просто набір V вузлів та набір пар вузлів E з V, що називаються шляхами (ребрами). Таким чином, граф – це спосіб подання зв'язків і відносин між парами об'єктів із деякої множини V. Маршрутом графа називається послідовність змінюваних вузлів і шляхів, що починається в одному і закінчується в іншому вузлі, причому кожен шлях є інцидентним щодо свого попередника і послідовника.

При пошуку шляху типовою проблемою є обхід перешкод. Найпростішим підходом до проблеми є ігнорування перешкод до зіткнення з ними. Все, що потрібно знати – це відносне положення об'єкта та його цілі і ознаку блокування перешкодою. До алгоритмів даного типу належить алгоритм трасування навколо перешкоди та алгоритм надійного трасування.

Наведені вище техніки обходу перешкод хоча й можуть виконати допустиму або навіть адекватну роботу, існують ситуації, в яких єдино розумний підхід – це планування всього шляху перед початком переміщень. Ці методи не можуть вирішити проблему зважених областей, яка полягає не стільки в обході перешкод, скільки у пошуку шляху з найменшою вартістю серед інших варіантів, де вартість місцевості може змінюватися. Проте, у теорії графів є кілька алгоритмів, які можуть вирішити проблему і складних перешкод, і зважених областей. У літературі, багато з цих алгоритмів подані в термінах зміни станів або проходження по вузлах графа. Застосування цих алгоритмів до пошуку шляху в геометричному просторі вимагає простої адаптації: стан або вузол графа представляють об'єкт, що знаходиться в певній клітинці, і пересування в сусідні клітинки відповідає переміщенню в сусідній стан або суміжні вузли. До основних алгоритмів даного типу належать алгоритми пошуку в глибину, ширину, алгоритм послідовних наближень при пошуку в глибину, алгоритм двонаправленого пошуку в ширину, алгоритм Дейкстри, алгоритм "кращий-перший" та алгоритм A*.

Алгоритм пошуку в глибину (DFS, depth-first search) в ненаправленому графі – це алгоритм, що використовується для ряду обчислень у графі, включаючи пошук шляху з одного вузла в інший (завдання трасування маршруту), визначення зв'язності графа і створення кістяка зв'язного графа. Обхід графа починаємо від деякого вузла s графа G. Вузол s тепер є поточним – назвемо його u. Потім рухаємося по G уздовж довільного шляху (u,v), інцидентного для поточного вузла u. Якщо шлях (u,v) веде до вже пройденого вузла v, негайно повертаємося в u. Якщо навпаки – (u,v) веде в новий вузол, рухаємося у вузол v і тепер він стає поточним, після чого вся процедура повторюється. При цьому можливе попадання в глухий кут, тобто з поточного вузла u всі шляхи, що пройшли через нього, будуть вести у вже пройдені вузли. Таким чином, вибір будь-якого з шляхів, що проходять через u, повертає назад в u. Щоб вирішити цю ситуацію, прямуємо в зворотному напрямку по шляху у вузол v, що передував вузлу u і який знову стає поточним вузлом. Далі рухаємося по будь-якому іншому шляху, що проходить через v і не використовувався. Якщо всі наскрізні шляхи v приводять у вже пройдені вузли, то знову повертаємося у вузол, що передує v, після чого повторюємо все заново. Таким чином продовжуємо рухатися в зворотному напрямку, поки не виявимо вузол, не всі шляхи якого пройдені, вибираємо один з них і прямуємо далі. Процес закінчується, якщо в результаті руху назад повертаємося в точку відправлення, не виявивши жодного непройденого наскрізного для

вузла s шляху. З точки зору часу виконання пошук в глибину є досить ефективним методом обходу графа. Відзначимо, що DFS викликається в кожному вузлі тільки один раз, а кожен шлях проходиться рівно двічі – по разу для кожної з його кінцевих точок. Порядок обходу графа за алгоритмом пошуку в глибину зображено на рисунку 1.

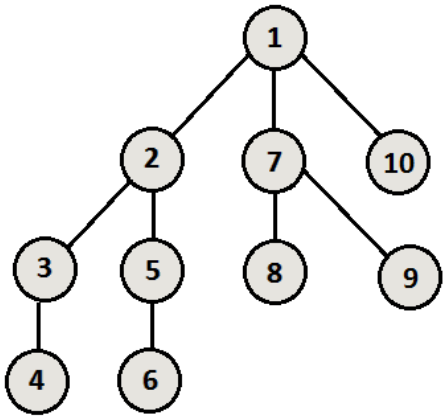

Рис.1. Порядок обходу графа в глибину

Алгоритм послідовних наближень при пошуку в глибину (IDDFS). У дійсності в алгоритмі пошуку у глибину існує ще одна проблема – вибір правильної глибини зупинки. Якщо вона буде дуже маленькою, то шлях не буде знайдений, якщо ж занадто великою, то можна витратити багато часу на марно або знайти шлях з дуже високою вартістю. Ці проблеми вирішуються ітеративним поглибленням – техніка, при якій виконується пошук в глибину зі зростаючою глибиною до тих пір, поки шлях не буде знайдений. При пошуку шляху ми можемо не починати з глибини, що рівна одиниці, а відразу взяти глибину, рівну відстані по прямій від старту до цілі.

Алгоритм пошуку в ширину (BFS, breadth-first search). Пошук в ширину більш передбачуваний, ніж пошук в глибину. Замість обходу лабіринту графа BFS виконує кругові обходи та розподіляє вузли за рівнями. Пошук у ширину починається з вузла s, розташованого на нульовому рівні. У цьому випадку заходимо в кожен суміжний з s вузол і відзначаємо його як пройдений. Тепер ці вузли утворюють перший рівень. При другому заході перевіряємо нові вузли, суміжні з вузлами першого рівня, але ті, що не ввійшли до нього. Ці вузли формують другий рівень і т.д. BFS-обхід закінчується тільки після обходу всіх вузлів графа. Порядок обходу графа за алгоритмом пошуку в ширину зображено на рисунку 2.

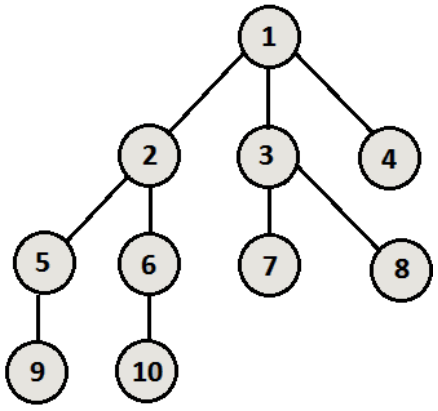

Рис.2. Порядок обходу графа в ширину

Алгоритм двонаправленого пошуку в ширину. У цьому методі запускаються два одночасних пошуки у ширину зі стартового і кінцевого вузлів та зупиняються, коли вузол з одного фронту пошуку знаходить сусідній вузол з іншого.

BFS-обхід може виконувати ті ж операції, що і DFS-обхід. Але між ними є і ряд відмінностей, зокрема, існують завдання, з якими кожен з методів справляється краще за інший. BFS-обхід набагато швидше визначає найкоротший маршрут у графі. DFS-обхід вирішує складні завдання взаємодії, наприклад, чи може кожна пара вузлів у графі бути пов'язаною двома несуміжними шляхами. Ці властивості, однак, застосовуються лише у ненаправлених графах. Тим

не менш і для направлених графів ці алгоритми можуть мати деяке застосування. BFS підхід має певний сенс у випадках, коли кожен шлях не є еквівалентним іншим шляхам. І навпаки, зустрічаються ситуації, в яких він просто не застосовний. Наприклад, граф можна використовувати для подання комп'ютерної мережі (наприклад, мережа Інтернет), і потрібно відшукати найкоротший шлях для пересилки пакету з однієї машини на іншу. У такому випадку пошук в ширину напевно не прийнятний, так як не всі шляхи мають однакову пропускну здатність (тобто в одній мережі може використовуватися оптико-волоконний зв'язок, а в іншій – малопотужна телефонна лінія). У цьому випадку визначено поняття зваженого графа.

Зваженим графом (weighted graph) називається граф, що має цифрове позначення w(e), що відповідає кожному шляху e і називається вагою (weight) шляху e.

Алгоритм Дейкстри (Dijkstra's algorithm) – алгоритм, що базується на застосуванні каскадного методу. Основна ідея його застосування в задачах визначення найкоротших маршрутів полягає у виконанні зваженого пошуку в ширину у вузлі v. Зокрема, використовується каскадний метод в алгоритмі, який ітеративно створює навколо вузла v «оточення» з вузлів, що входять у цю множину, в порядку їх відстаней від v, вибираючи при кожній ітерації вузол, найближчий до v. Алгоритм закінчує роботу, як тільки вузли закінчуються, тобто визначені найкоротші шляхи з v до всіх інших вузлів в G. Незважаючи на простоту, цей підхід дуже ефективний.

Алгоритм «кращий-перший». Це перший розглядуваний евристичний пошук, який бере до уваги знання про простір пошуку для направлення своїх зусиль. Він схожий на алгоритм Дейкстри, за винятком того, що вузли в списку Open оцінюються за приблизною відстанню, що залишилася до мети. Ця оцінка так само не вимагає наявності оновлень, на відміну від алгоритму Дейкстри.

Алгоритм A* – це алгоритм пошуку по першому найкращому співпадінні на графі, який знаходить маршрут з найменшою вартістю від однієї вершини (початкової) до іншої (цільової, кінцевої). Алгоритм поєднує в собі врахування довжини попереднього шляху з алгоритму Дейкстри з евристикою з алгоритму «кращий-перший».

Порядок обходу вершин визначається евристичною функцією «відстань + вартість» (зазвичай позначається як $f(x)$). Ця функція – сума двох інших: функції вартості досягнення даної вершини (x) з початковою (зазвичай позначається як $g(x)$ і може бути евристичною) і евристичною оцінкою відстані від розглянутої вершини до кінцевої (позначається як $h(x)$). Функція $h(x)$ повинна бути допустимою евристичною оцінкою, тобто не повинна переоцінювати відстані до цільової вершини. Наприклад, для задачі маршрутизації $h(x)$ може представляти собою відстань до мети по прямій лінії, тому що це фізично найменша можлива відстань між двома точками.

На початку роботи проглядаються вузли, суміжні з початковим вузлом; обирається той з них, який має мінімальне значення $f(x)$, після чого цей вузол розкривається. На кожному етапі алгоритм оперує з множиною шляхів з початкової точки до всіх ще не розкритих (листових) вершин графа, яка розміщується у черзі з пріоритетом. Пріоритет шляху визначається за значенням $f(x)=g(x)+h(x)$. Алгоритм продовжує свою роботу до тих пір, поки значення $f(x)$ цільової вершини не виявиться меншим, ніж будь-яке значення в черзі (або поки все дерево не буде переглянуто). З множинних рішень вибирається рішення з найменшою вартістю.

Загалом кажучи, пошук в глибину і пошук у ширину є двома окремими випадками алгоритму A*. Для пошуку в глибину візьмемо глобальну змінну-лічильник C, ініціалізувавши її деякими великим значенням. Кожного разу при розкритті вершини будемо присвоювати значення лічильника суміжних вершин, зменшуючи його на одиницю після кожного присвоєння. Таким чином, чим раніше буде відкрита вершина, тим більше значення $h(x)$ вона отримає, а значить, буде переглянута в останню чергу. Якщо покласти $h(x)=0$ для всіх вершин, то ми отримаємо ще один спеціальний випадок – алгоритм Дейкстри.

Передбачено декілька особливостей реалізації і прийомів, які можуть значно вплинути на ефективність алгоритму A*. Перше, на що треба звернути увагу – це те, як черга з пріоритетом обробляє зв'язок між вершинами. Якщо вершини додаються до неї так, що черга працює за принципом LIFO, то у випадку вершин з однаковою оцінкою A* «піде» в глибину. Якщо ж при додаванні вершин реалізується принцип FIFO, то для вершин з однаковою оцінкою алгоритм, навпаки, буде реалізовувати пошук в ширину. У деяких випадках ця обставина може надавати істотне значення на продуктивність.

Існує і безліч інших алгоритмів, що в основному є модифікацією вище розглянутих і спрямовані на те, що мінімізувати недоліки основних алгоритмів: зменшення витрат пам'яті, підвищення швидкодії алгоритму, вибір оптимальних параметрів для його реалізації. До таких

алгоритмів належать алгоритм A* з ітеративним поглибленням (iterative deeping A*, IDA*), A* з обмеженням пам'яті (memory-bounded A*, MA*), спрощений MA* (simplified MA*, SMA*), рекурсивний пошук по першому найкращому збігу (recursive best-first search, RBFS), алгоритм Белмана-Форда, алгоритм Флойда-Уоршелла, алгоритм Джонсона тощо.

Проаналізувавши принципи роботи кожного з алгоритмів, можна виділити недоліки та переваги їх застосування для задачі трасування маршруту. Проблеми методу переміщення у випадковому напрямку виникають, якщо перешкоди великі або увігнуті – об'єкт може повністю застрягти або як мінімум втратити багато часу, поки не буде знайдений обхідний шлях. Проблема у методі трасування навколо перешкоди полягає в ухваленні рішення – коли ж припинити трасування. При неправильному прийнятті цього рішення можна постійно кружляти всередині перешкоди не знаходячи шляхи назовні. Підхід надійного трасування часто використовує набагато більше часу на трасування ніж необхідно. Компроміс полягає в комбінації обох підходів: завжди використовувати евристику простіше для зупинки трасування, але якщо зафіксовано зациклення, перемикатися на надійну евристику. Метод пошуку в ширину знаходить шлях навколо перешкод, і цей шлях є найкоротшим, тобто одним з декількох найкоротших у довжину шляхів, якщо всі кроки мають однакову вартість. Тут є безліч простих проблем. Одна з них полягає в тому, що пошук йде рівномірно у всіх напрямках, замість того, щоб бути спрямованим у бік цілі. Інша ж проблема полягає в тому, що не всі кроки рівні, принаймні, кроки по діагоналі повинні бути довшими ортогональних. Двонаправлений пошук в ширину може поліпшити простий пошук у ширину (зазвичай, у 2 рази), але все ще є дуже неефективним. Алгоритм Дейкстри має дві переваги в порівнянні з пошуком у ширину: він бере до уваги вартість або довжину шляху і оновлює вузли, якщо до них знайдений кращий шлях. Хоча використання методу Дейкстри для вирішення проблем графів не завжди ефективне (неможливість працювати з від'ємною вагою), тим не менше існує область (так зване «завдання комівояжера»), в якій його застосування дає найкраще вирішення проблеми. Однак, він має недолік пошуку в ширину, ігноруючи напрямок до цілі. Алгоритм послідовних наближень при пошуку в глибину є асимптотично оптимальним по часу і пам'яті серед усіх алгоритмів, що використовують перебір. Найшвидшим з усіх розглянутих алгоритмів, що виконують планування, є алгоритм «кращий-перший», який направляється по прямій до цілі. Він так само має і свої недоліки. Він може не брати до уваги накопичену вартість шляху, прямуючи по прямій через зону з високою вартістю, а не обходячи її.

Найкращим алгоритмом для пошуку оптимальних шляхів у різних просторах є алгоритм A*. Даний алгоритм має безліч цікавих властивостей. Він гарантовано знаходить найкоротший шлях, до тих пір, поки евристичне наближення $h(n)$ є допустимим, тобто він ніколи не перевищує дійсної відстані, що залишилася до цілі. Цей алгоритм найкращим чином використовує евристику. Алгоритм A* набагато краще справляється з ситуаціями, проблемними для інших алгоритмів і на практиці виявляється дуже гнучким.

Список використаних джерел

1. http://www.codenet.ru/progr/other/prbook/gl9.php
2. http://www.policyalmanac.org/games/aStarTutorial_rus.htm
3. http://algolist.manual.ru/maths/graphs/search.php
4. Евстигнеев В.А. Применение теории графов в программировании. – М.: Наука, 1985. – 352 с.
5. Зыков А.А. Основы теории графов. – М.: Наука, 1987. – 384 с.
6. Ю.В. Нікольський, В.В. Пасічник, Ю.М. Щербина. Дискретна математика: Підручник. – Львів: Магнолія - 2006, 2009. – 432 с.
7. Магрупов Т.М. Графы, сети, алгоритмы и их приложения. – Ташкент: Фан, 1990. – 120 с.
8. Свами М., Тхуласираман К. Графы, сети и алгоритмы. – М.: Мир, 1984. – 454 с.
9. Берж К. Теория графов и ее применения. – М.: ИЛ, 1962. – 320 с.