Крім цього, метод пошуку із поверненням дозволяє вирішувати безліч інших перебірних завдань. Наприклад, за допомогою нього можна отримати всі підмножини, розміщення, перестановки, поєднання даної множини М.

Метод пошуку із поверненням є універсальним. Досить легко проектувати та програмувати алгоритми розв'язання задач з використанням цього методу. Але оскільки пошук з поверненням виконується послідовно, його часова складність залежить від розміру набору даних. Час знаходження рішення може бути дуже велике навіть при невеликих розмірностях завдання (кількості вихідних даних), причому настільки великий (може становити роки або навіть століття), що про практичне застосування не може бути мови.

Тим не менш, пошук з поверненням має свої використання. Цей метод може бути корисним у випадках, коли даних небагато, коли точна позиція шуканого елемента не відома або як пошук на випадок, коли інші методи пошуку недоречні.

Список літератури

1. *Алгоритм пошуку з поверненням, URL:*http://um.co.ua/9/9-10/9-106797.html
2. *Пошук з вертанням, URL:* https://www.wik.uk-ua.nina.az.html
3. *Рекурсивні дерева: Огляд, URL: https://probability.knu.ua/tims/issues-new/51/PDF/3.pdf*

УДК 519.1

Чемес В.С., студент 1 курсу
Спеціальність 122 «Комп'ютерні науки»
Факультет інформаційних та прикладних технологій
Ніколюк П.К. д-р фіз-мат наук
кафедри інформаційних технологій

ВИКОРИСТАННЯ ГРАФІВ ДЛЯ ГЕНЕРУВАННЯ ЛАБІРИНТІВ

Донецький національний університет імені Василя Стуса, м. Вінниця

Перед тим як створювати алгоритми для генерування лабіринтів, потрібно спершу дати визначення лабіринту. Лабіринт - це складний систематично побудований лабіринт або сплутаний шлях, який часто використовується як інструмент для розваг або розвитку когнітивних навичок. Лабіринти можуть мати різні форми та розміри, але вони завжди складаються з декількох вузьких коридорів, які переплітаються та розводяться, часто з безвихідними галереями, що створюють складний шлях до цілі або виходу. Лабіринти зазвичай використовуються як інструмент для розвитку когнітивних навичок, таких як концентрація, логіка, пам'ять та розв'язання проблем. Їх також можна

використовувати як інструмент для розваг, наприклад, у парках атракціонів та ігрових майданчиках.

У програмному значенні лабіринт може відноситися до різноманітних програм або алгоритмів, які містять складну структуру розгалужень або повторень, унаслідок чого їх виконання може бути складним і заплутаним, як у лабіринті. Наприклад, у програмуванні це можуть бути складні алгоритми розгалуження (наприклад, рішення задачі комівояжера), які містять велику кількість умовних операторів і циклів. Ці алгоритми можуть бути складними для розуміння і налагодження, якщо не враховувати їх структуру та взаємодії між різними частинами коду. Крім того, термін "лабіринт" може використовуватися для опису складних програм, які містять безліч взаємодіючих модулів, класів і функцій. У таких програмах важко знайти шлях від початку до кінця, оскільки код може мати багато рівнів абстракції та складних взаємодій між різними частинами [1].

Лабіринти можуть бути створені з визначеним розташуванням клітин, найчастіше в формі прямокутної сітки зі стінками між ними. Це можна розглядати як зв'язний граф з ребрами, що представляють стінки, і вузлами, що представляють клітини. Головна мета алгоритму створення лабіринту - це створення підграфу, в якому важко знайти шлях між двома окремими вузлами. Якщо підграф не є зв'язним графом, то існують непотрібні області графу, які не сприяють простору пошуку. Можуть бути кілька шляхів між вибраними вузлами, якщо граф містить петлі. Тому для створення лабіринту часто використовують генерацію випадкового кістякового дерева. Петлі можуть бути введені до результату під час виконання алгоритму, додавши випадкові ребра.

Для генерування лабіринту використовуються різні алгоритми. Перший, це алгоритм пошуку в глиб. Алгоритм пошуку в глиб (DFS - Depth-First Search) може бути використаний для генерації лабіринтів шляхом рекурсивного просування вглиб, позначаючи прохідні шляхи як відвідані. Наприклад, можна почати зі стартової клітинки та рекурсивно просуватись в один з сусідніх прохідних шляхів. Якщо немає можливості рухатись далі, повертаються назад до попередньої клітинки та продовжують рухатись в іншому напрямку. Клітинки, які були відвідані, позначаються як зайняті. Цей процес триває доти, доки не буде знайдений вихід або не будуть пройдені всі доступні клітинки. Отриманий лабіринт може бути додатково випробуваний на ефективність вирішення задач знаходження шляху від старту до фінішу, щоб перевірити, чи є він дійсно складним та цікавим для гравців [2].

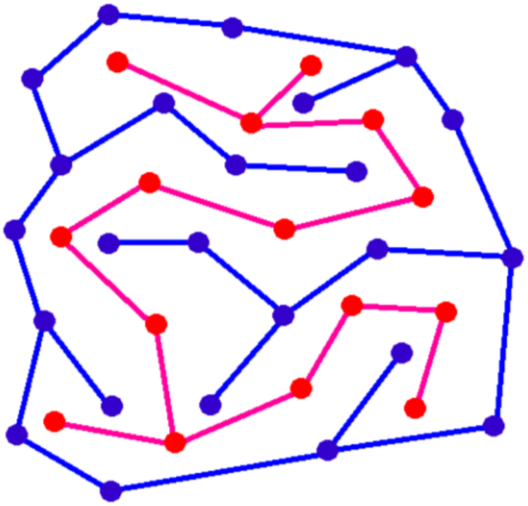

Рис. 1. Результат роботи алгоритму пошуку в глиб

Ось код алгоритму пошуку в глибину на мові програмування Python

```
import random

def generate_maze(width, height):
    maze = [['#' for x in range(width)] for y in range(height)]
    start_x = random.randint(0, width-1)
    start_y = random.randint(0, height-1)
    maze[start_y][start_x] = 'S'
    dfs(maze, start_x, start_y)
    end_x = random.randint(0, width-1)
    end_y = random.randint(0, height-1)
    while maze[end_y][end_x] != ' ':
        end_x = random.randint(0, width-1)
        end_y = random.randint(0, height-1)
    maze[end_y][end_x] = 'E'
    return maze

def dfs(maze, x, y):
    neighbors = get_neighbors(maze, x, y)
    if not neighbors:
        return
    next_x, next_y = random.choice(neighbors)
    if next_x < x:
        maze[y][x-1] = ' '
    elif next_x > x:
        maze[y][x+1] = ' '
    elif next_y < y:
        maze[y-1][x] = ' '
```

```
        else:
            maze[y+1][x] = ' '
        dfs(maze, next_x, next_y)

    def get_neighbors(maze, x, y):
        neighbors = []
        if x > 1 and maze[y][x-2] == '#':
            neighbors.append((x-2, y))
        if x < len(maze[0])-2 and maze[y][x+2] == '#':
            neighbors.append((x+2, y))
        if y > 1 and maze[y-2][x] == '#':
            neighbors.append((x, y-2))
        if y < len(maze)-2 and maze[y+2][x] == '#':
            neighbors.append((x, y+2))
        return neighbors
```

Алгоритм Прима є одним з алгоритмів генерації лабіринтів, який базується на побудові мінімального каркасного дерева. Його можна використовувати для створення лабіринтів, де з'єднання комірок поля здійснюється по вертикалях та горизонталях. Для використання алгоритму Прима для генерації лабіринту, необхідно спочатку створити сітку комірок, які будуть представляти стіни лабіринту. Далі, обирається випадкова комірка та додається до множини відвіданих комірок. Потім, використовуючи ваги ребер між комірками, що з'єднують їх по вертикалях та горизонталях, обирається ребро, що має найменшу вагу та з'єднує відвідану та невідвідану комірки. Ребро додається до множини ребер лабіринту та невідвідана комірка додається до множини відвіданих комірок. Цей процес повторюється до тих пір, поки всі комірки лабіринту не будуть відвідані. У кінцевому результаті має бути побудовано дерево з'єднання комірок, яке є мінімальним за вагою, тобто мінімальним каркасним деревом. Це дерево та його ребра відповідають стінам лабіринту. Для отримання готового лабіринту необхідно замінити дерево на стіни, збережені в множині ребер лабіринту, що не входять до мінімального каркасного дерева(Рис. 1) [3] .

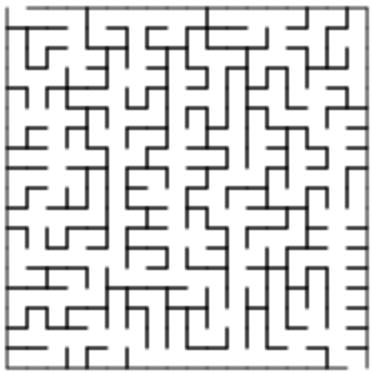

Рис. 2. Результат роботи алгоритму Прима

Алгоритм Ейлера - це алгоритм пошуку шляху в графі, що проходить через кожну вершину графу лише один раз. Застосування цього алгоритму для генерації лабіринту передбачає створення графу, де кожна вершина відповідає комірці лабіринту, а кожний ребро відповідає стіні між комірками.

Для того, щоб застосувати алгоритм Ейлера для генерації лабіринту, потрібно виконати ряд певних операцій:

Створити сітку комірок, в якій буде зберігатися лабіринт. Кожна комірка має чотири стіни: верхню, нижню, ліву та праву. Взяти будь-яку комірку в лабіринті та відмітити її як поточну. Знайти всі сусідні комірки, які ще не були відвідані та видалити стіну між поточною коміркою та випадковою сусідньою коміркою. Це створить прохід між двома комірками в лабіринті. Потім потрібно відмітити сусідню комірку, з якою було створено прохід, як нову поточну комірку та зробити рекурсію, доки всі комірки в лабіринті не будуть відвідані. Цей процес буде продовжуватися до тих пір, поки не буде створено прохід між кожною коміркою в лабіринті, і буде створено повний лабіринт [4].

Отже, лабіринт – це системний сплутаний шлях, який можна вважати графом для подальшої роботи з ним. Для генерування лабіринтів будь-якого розміру та рівня складності можна використовувати графи та алгоритми, які базуються на роботі з ними.

Список літератури:

1. *Baelbung “Algorithm to Generate a Maze”, URL:* https://www.baeldung.com/cs/maze-generation
2. *Geek for geeks “Depth First Search or DFS for a Graph”, URL:* https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
3. *Wikipedia “Prim's algorithm”, URL:* https://en.wikipedia.org/wiki/Prim%27s_algorithm
4. *The Buckblog “Maze Generation: Eller's Algorithm”, URL:* https://weblog.jamisbuck.org/2010/12/29/maze-generation-eller-s-algorithm

УДК 004.9

Шафорост В. В.,
студент 3 курсу спеціальності 122
«Комп’ютерні науки»
Січко Т. В., к. т. н., доцент, доцент
кафедри інформаційних технологій

АЛГОРИТМІЗАЦІЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Донецький національний університет імені Василя Стуса

Розробка програмного забезпечення та створення алгоритмів є важливою частиною усіх готових програмних продуктів.