DOI: https://doi.org/10.15276/ict.02.2025.25
УДК 519.2:004.42

Модифікація використання алгоритму Флойда-Уоршелла при роботі з графами деревної структури

Дрозд Юлія Володимирівна[1)]
Канд. техніч. наук, доцент каф. Інформаційних систем
ORCID: https://orcid.org/0009-0009-9151-4747; yuliia.drozd@op.edu.ua. Scopus Author ID: 56007618700
Дрозд Мирослав Олександрович[1)]
Канд. техніч. наук, доцент каф. Іінформаційних систем
ORCID: https://orcid.org/0000-0003-0770-6295; myroslav.drozd@op.edu.ua. Scopus Author ID: 56667174000
Шпинковський Олександр Анатолійович[1)]
Канд. техніч. наук, доцент каф. Інформаційних систем
ORCID: https://orcid.org/0000-0002-7000-0327; shpinkovski@op.edu.ua. Scopus Author ID: 57060560200
Шпинковська Марія Іванівна[1)]
Канд. техніч. наук, доцент каф. Вищої математики та моделювання систем
Scopus Author ID: 57060466800
[1)] Національний університет «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна

АНОТАЦІЯ

Невід'ємною частиною будь-якого програмного проекту є використання алгоритмів. Алгоритми, які є канвою, основою великої кількості програмних рішень є класичними, надрукованими у настільних книжках програміста та ще часто-густо реалізовані на багатьох мовах програмування. Але будь-який програмний проект має багато тонкощів, які неможливо вичерпати класичним підходом, пошуки вже готових рішень за заданими параметрами також бувають невдалі, або готові функції не відповідають заданим критеріям швидкодії або точності. Прикладом такого алгоритму, розв'язання якого запропоновано в даній роботі є алгоритм пошуку всіх мінімальних відстаней між будь-якими двома верхівками на графі деревної структури. За основу його написання було використано алгоритм Флойда-Уоршелла, універсальний, створений для графів будь-якої структури, циклічних або орієнтованих. В роботі доведено, що деревна структура графа значно простіша ніж узагальнена циклічна структура, де у якості зв'язків можуть бути ребра або можуть бути дуги. В роботі показано, що у дереві існує лише один шлях між будь якими двома верхівками, тобто ми не маємо жодної необхідності перераховувати отриману відповідь – відстань між двома певними верхівками, сподіваючись, що вона буде більш оптимальною згідно критеріям завдання, отже ми можемо позбутися величезної кількості умовних операторів та зовнішнього циклу, кількість ітерацій якого відповідає кількості верхівок, тобто, якщо у звичайному проекті кількість верхівок графа деревної структури стартує від тисячі верхівок, то використовуючи модифікований алгоритм Флойда спеціально під графи деревної структури, а не розповсюджену, так звану, універсальну версію, можна пришвидшити роботу програмного продукту щонайменше вдвічі. Крім того, графи деревної структури мають ще низку структурних відмінностей, які дозволяють прискорити ще від двох до десяти разів по відношенню до готової к використанню універсальної версії алгоритму. Отже, в роботі розглянута методика пристосування класичного алгоритму до специфічних потреб, якими є неорієнтовані графи деревної структури.

Ключові слова: алгоритми на графах; пошук мінімальної відстані; оптимізація розрахунків; вплив умовного оператора на швидкодію програм; алгоритм Флойда-Уоршелла; графи деревної структури

Актуальність. З розвитком різноманітних технологій обробки даних та переходу на автоматизовану обробку великих масивів даних паралельно крокують технології побудови цифрових схем, відповідно, зростає швидкодія процесорів та об'єми оперативної пам'яті, але, нажаль, зростання характеристик, так званого, заліза майже завжди на крок чи два відстає від вимог сучасного програмного забезпечення. Відповідно, питання оптимізації розрахунків з точки зору вимог до оперативної пам'яті або пришвидшення за рахунок виключення дублюючих операцій, за рахунок виключення умовних операторів і, навіть, розгортання циклів з кінцевою кількістю ітерацій, є актуальними зараз і будуть актуальними у найближчому майбутньому. Зокрема, в запропонованій роботі ми пришвидшуємо обробку інформації за рахунок виключення зайвих дій, які в принципі притаманні ітераційному підходу.

Метою дослідження є порівняння класичного алгоритму Флойда-Уоршелла щодо визначення мінімальної відстані між будь-якими двома верхівками на графі довільної

структури та його оптимізованого варіанту, пристосованого для графів деревної структури. В основі дослідження лежить порівняння загальної кількості операцій та загальної кількості викликів умовного оператора між класичним алгоритмом Флойда-Уоршелла та його оптимізованим аналогом.

Класичний алгоритм Флойда-Уоршелла працює з двома матрицями: матрицею відстаней та матрицею шляхів, загальна кількість операцій представляє собою кубічну залежність від кількості верхівок на графі і дорівнює, відповідно, 2×N×N×N, де N – кількість верхівок графа. Серія таблиць нижче ілюструє роботу алгоритму Флойда-Уоршелла для графа з 5 верхівок (рис. 1), який має орієнтовані та неорієнтовані зв'язки та містить цикли. На таблицях 1-5 продемонстроване покрокове перетворення матриці суміжності у матрицю відстаней та матриці прямих шляхів у матрицю прямих та транзитних шляхів, огляд того, як перетворюються дві робочі структури алгоритму: матриця суміжності та матриця шляхів, дозволяє досить швидко написати робочий код програми та, відповідно, досить швидко оцінити складність її роботи, а також локалізувати моменти, які можна взагалі прибрати для аналогічних розрахунків графа деревної структури, тому що дерева – є більш простою структурою, зокрема, там не існує альтернативних шляхів між двома верхівками на відміну від графа з універсальною структурою.

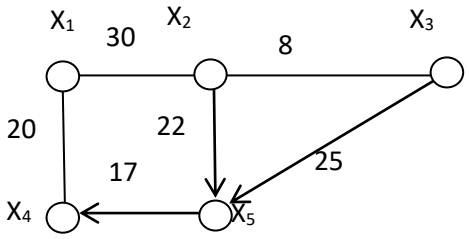

Рис.1. **Граф для алгоритму Флойда-Уоршелла**

Таблиця 1. **Матриця суміжності і матриця шляхів для алгоритму Флойда-Уоршелла на старті роботи програми**

	X_1	X_2	X_3	X_4	X_5		1	2	3	4	5
X_1	-	30	∞	20	∞	1	-	2	3	4	5
X_2	30	-	8	∞	22	2	1	-	3	4	5
X_3	∞	8	-	∞	25	3	1	2	-	4	5
X_4	20	∞	∞	-	∞	4	1	2	3	-	5
X_5	∞	∞	∞	17	-	5	1	2	3	4	-

Таблиця 2. **Матриця відстаней і матриця шляхів для алгоритму Флойда-Уоршелла після першої ітерації**

	X_1	X_2	X_3	X_4	X_5		1	2	3	4	5
X_1	-	30	∞	20	∞	1	-	2	3	4	5
X_2	30	-	8	**50**	22	2	1	-	3	1	5
X_3	∞	8	-	∞	25	3	1	2	-	4	5
X_4	20	**50**	∞	-	∞	4	1	1	3	-	5
X_5	∞	∞	∞	17	-	5	1	2	3	4	-

Таблиця 3. **Матриця відстаней і матриця шляхів для алгоритму Флойда-Уоршелла після другої ітерації**

	X_1	X_2	X_3	X_4	X_5		1	2	3	4	5
X_1	-	30	**38**	20	**52**	1	-	2	2	4	2
X_2	30	-	8	50	22	2	1	-	3	1	5
X_3	**38**	8	-	**58**	25	3	2	2	-	2	5
X_4	20	50	**58**	-	**72**	4	1	1	2	-	2
X_5	∞	∞	∞	17	-	5	1	2	3	4	-

Таблиця 4. **Матриця відстаней і матриця шляхів для алгоритму Флойда-Уоршелла після третьої ітерації**

	X_1	X_2	X_3	X_4	X_5		1	2	3	4	5
X_1	-	30	38	20	52	1	-	2	2	4	2
X_2	30	-	8	50	22	2	1	-	3	1	5
X_3	38	8	-	58	25	3	2	2	-	2	5
X_4	20	50	58	-	72	4	1	1	2	-	2
X_5	∞	∞	∞	17	-	5	1	2	3	4	-

Таблиця 5. **Матриця відстаней і матриця шляхів для алгоритму Флойда-Уоршелла після закінчення роботи програми**

	X_1	X_2	X_3	X_4	X_5		1	2	3	4	5
X_1	-	30	38	20	52	1	-	2	2	4	2
X_2	30	-	8	50	22	2	1	-	3	1	5
X_3	38	8	-	58	25	3	2	2	-	2	5
X_4	20	50	58	-	72	4	1	1	2	-	2
X_5	**37**	**67**	**75**	17	-	5	4	4	4	4	-

По цих таблицях напишемо базовий фрагмент коду програми на мові програмування С, а потім проведемо його аналіз щодо оптимізації та скорочення у відповідності із задекларованими цілями. Фрагмент коду програми із коментарями виглядає наступним чином:

```
for (k=0; k<n; k ++){
for (i=0; i<0; i++){
      for (j=0; j<0; j++){
          if ((i!=k)&&(j!=k)&&(A[i][k]!=0)&&(A[k][j]!=0)) {
                change=A[i][k]+A[k][j];
                if(change<A[i][j]){
                   A[i][j]= change;  B[i][j]= k; }}}}}
```

Ось так виглядає програмний код, складність оцінюємо по кількості викликів найдовшої програмної операції – умовного виклику, тобто оператора if. Їх в даній програмі буде 2×N×N×N разів, що буквально видно з тексту програми. Перший умовний оператор перевіряє, чи взагалі поточна верхівка спроможна бути транзитом між певними двома, а наступний умовний оператор оцінює, чи він дешевший ніж попередній отриманий результат.

Тепер проведемо аналіз аналогічних обчислень на графі деревної структури. Для прикладу візьмемо граф-дерево з середньою кількістю кінцевих верхівок: 5 від 8, де 8 – загальна кількість верхівок графа. Кінцеві верхівки не можуть бути транзитними за рахунок чого кількість зовнішніх ітерацій значно зменшується на графах деревної структури. Наприклад, на графі на рис. 2 транзитними можуть бути лише верхівки 6, 7 та 8. На першому кроці транзитною призначаємо верхівку 6, відповідно всі розрахунки проводимо відносно цієї верхівки (таблиця 6). Наступна пара таблиць ілюструє результати цих розрахунків.

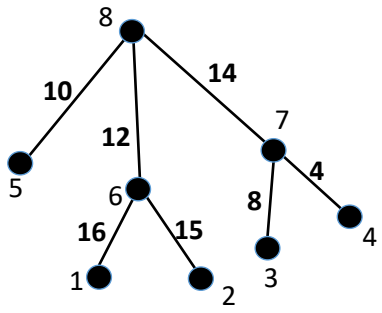

Рис.2. **Граф деревної структури**

Таблиця 6. **Матриці відстаней і шляхів для графу деревної структури на старті роботи програми**

	1	2	3	4	5	6	7	8
1	-					16		
2		-				15		
3			-				8	
4				-			4	
5					-			10
6	16	15				-		12
7			8	4			-	14
8					10	12	14	-

	1	2	3	4	5	6	7	8
1	-					6		
2		-				6		
3			-				7	
4				-			7	
5					-			8
6	1	2				-		8
7			3	4			-	8
8					5	6	7	-

Першою транзитною верхівкою була шоста, другою – сьома. На останньому кроці транзитною стає верхівка з номером вісім. Відповідно програма обчислює всі відстані відносно неї, не змінюючи вміст комірок восьмого стовпчика та восьмого рядочка. Результат розрахунків виглядає наступним чином:

Таблиця 7. **Матриці відстаней і шляхів для графу деревної структури після останньої ітерації**

	1	2	3	4	5	6	7	8
1	-	31	50	46	38	16	42	28
2	31	-	49	45	37	15	41	27
3	50	49	-	12	32	34	8	22
4	46	45	12	-	28	30	4	18
5	38	37	32	28	-	22	24	10
6	16	15	34	30	22	-	26	12
7	42	41	8	4	24	26	-	14
8	28	27	22	18	10	12	14	-

	1	2	3	4	5	6	7	8
1	-	6	8	8	8	6	8	6
2	6	-	8	8	8	6	8	6
3	8	8	-	7	8	8	7	7
4	8	8	7	-	8	8	7	7
5	8	8	8	8	-	8	8	8
6	1	2	8	8	8	-	8	8
7	8	8	3	4	8	8	-	8
8	6	6	7	7	5	6	7	-

Нарешті ми отримали повністю заповнену матрицю відстаней та матрицю шляхів. Щонайменше скорочення роботи програми відносно аналогічного графу циклічної структури вдвічі відбулось за рахунок невеликої кількості транзитних верхівок, їх вдвічі менше від загальної кількості верхівок на розглянутому графі деревної структури, але такі узагальнення, нажаль, неможливі, тому що кількість кінцевих верхівок на графах деревної структури варіюється від двох до N-1, де N – загальна кількість верхівок.

Тепер перелічимо кроки оптимізації даної програми відносно графів деревної структури. На рис. 2 ми бачимо граф деревної структури, а в таблицях 6-7 ми бачимо обробку цього графу модифікованим алгоритмом Флойда-Уоршелла.

По цих таблицях напишемо базовий фрагмент коду програми на мові програмування С, а потім проведемо його аналіз щодо оптимізації та скорочення у відповідності із задекларованими цілями. Отже фрагмент коду програми із коментарями виглядає наступним чином:

```
for (k=0; k<n; k ++){
    for (i=0; i<0; i++){
        for (j=0; j<0; j++){
            if ((i!=k)&&(j!=k)&&(A[i][k]!=0)&&(A[k][j]!=0)&& (A[i][j]= =0)){
                    A[i][j])=A[i][k]+A[k][j];  B[i][j]= k; }}}}
```

Отже, як бачимо, у нас в оптимізованій версії програми лише один умовний оператор, це пов’язано із тим, що в графах деревної структури існує лише один шлях між двома верхівками, якщо ми його отримали один раз, то більше не шукаємо, відповідно, немає ніяких порівнянь і пов’язаних із ними викликів умовних операторів. Але той самий умовний оператор знаходиться всередині самого внутрішнього циклу. Тобто має бути викликаний N×N×N раз. В разі його відсутності час виконання програми скорочується, відповідно, в два чи більше разів, тому що там ми позбавляємось ще від низки операторів присвоєння.

Наступна можливість оптимізації полягає в тому, що ми можемо працювати лише з верхньою половиною матриці, оскільки в неорієнтованому графі матриця суміжності симетрична відносно головної діагоналі. А надалі побудована на її підставі матриця відстаней симетрична відносно головної діагоналі, тобто за рахунок цього загальну кількість операцій ми ще раз скорочуємо вдвічі, відповідно при виконанні лише двох перелічених покращень, загальний час виконання програми скорочується в чотири рази.

Третім інструментом покращення даної програми може бути урахування кінцевих верхівок, тобто верхівок, степінь яких дорівнює одиниці. Кількість таких верхівок на графах деревної структури варіюється від двох, якщо граф має ланцюгову структуру, до N-1, якщо граф має структуру зірочки, але в середньому, кінцевих верхівок на графі деревної структури вдвічі менше ніж їх загальна кількість. Ці верхівки взагалі не можуть бути транзитними за визначенням, тому зовнішній цикл, відповідно має, скоротитися в середньому вдвічі, а загальний час роботи програми, відповідно, в вісім разів.

Четвертим інструментом пришвидшення даної програми є можливість в певних випадках взагалі проігнорувати матрицю шляхів, якщо вона неінформативна в певних умовах. Наприклад, нам необхідні лише самі числа щодо відстаней мінімальних, а певний шлях в умовах конкретної задачі може бути непотрібен.

Подальші практичні дослідження для графів різного розміру, але таких, що відповідають нагальним програмним продуктам, тобто від трьохсот до тисячі верхівок, мають підтвердити вищевикладені розрахунки.

Висновки. В роботі запропоновано чотири інструменти підвищення ефективності алгоритму Флойда-Уоршелла щодо графів деревної структури. За попередніми розрахунками час виконання програми при використанні заявленого підходу порівняно із класичним має

скоротитися в вісім-десять разів, це очевидно із буквально скороченого коду програми, але все одно потребує експериментів для графів із різною кількістю верхівок та різних конфігурацій.

СПИСОК ЛІТЕРАТУРИ

1. Yamane Y, Kobayashi K. “A New Fast Unweighted All-pairs Shortest Path Search Algorithm Based on Pruning by Shortest Path Trees (PST)”. *Procedia Computer Science*. 2019; 159: 2409–2418. DOI: https://doi.org/10.1016/j.procs.2019.09.418.

2. Planken L. R., de Weerdt M., van der Krogt R. P. J. “Computing All-Pairs Shortest Paths by Leveraging Low Treewidth”. *Journal of Artificial Intelligence Research*. 2014.

3. Planken L. R., de Weerdt M., van der Krogt R. P. J. “Computing All-Pairs Shortest Paths by Leveraging Low Treewidth”. *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*. 2011. P.170–177.

4. Yamane Y., Kobayashi K. “A New Fast Weighted All-pairs Shortest Path Search Algorithm Based on Pruning by Shortest Path Trees”. *arXiv*. 2019. DOI: https://doi.org/10.48550/arXiv.1908.06798.

5. Peng W., Hu X., Zhao F., Su J. “A Fast Algorithm to Find All-Pairs Shortest Paths in Complex Networks”. *Procedia Computer Science*. 2012; 9: 557–566. DOI: https://doi.org/10.1016/j.procs.2012.04.060.

6. Jing Z. “A fast algorithm for All-Pairs-Shortest-Paths suitable …” *PMC*. 2024.

7. Lancia G., Pennisi M., Pulvirenti A., & Rahmann S. “Speeding Up Floyd–Warshall’s Algorithm to Compute All-Pairs Shortest Paths”. *Algorithms.* 2025; 18 (4): 160. DOI: https://doi.org/10.3390/a18040160.

8. “Floyd–Warshall algorithm”. *Wikipedia*. 2025. – URL: https://en.wikipedia.org/wiki/Floyd–Warshall_algorithm.

DOI: https://doi.org/10.15276/ict.02.2025.25
UDC 519.2:004.42

Modification the use of the Floyd-Warshell algorithm when working with tree-structure graphs

Yuliia V. Drozd[1)]
PhD, Associate Professor of the Department of Information Systems
ORCID: https://orcid.org/0009-0009-9151-4747; yuliia.drozd@op.edu.ua. Scopus Author ID: 56007618700
Myroslav O. Drozd[1)]
PhD, Associate Professor of the Department of Information Systems
ORCID: https://orcid.org/0000-0003-0770-6295; myroslav.drozd@op.edu.ua. Scopus Author ID: 56667174000
Oleksandr A. Shpinkovski[1)]
PhD, Associate Professor of the Department of Information Systems
ORCID: https://orcid.org/0000-0002-7000-0327; shpinkovski@op.edu.ua
Maria I. Shpinkovska[1)]
PhD, Associate Professor of the Department of Higher Mathematics and Systems Modeling
Scopus Author ID: 57060466800
[1)] Odesa Polytechnic National University, 1, Shevchenko Ave. Odesa, 65044, Ukraine

ABSTRACT

In modern blockchain systems, smart contracts are one of the most critical components for ensuring the automated execution of agreements without the need for intermediaries. However, smart contracts written in languages like Solidity may contain vulnerabilities that can be exploited by malicious actors to steal funds or manipulate assets. Given the increasing number of attacks on smart contracts, the development of effective methods for detecting such vulnerabilities is crucial. Traditional approaches to detecting vulnerabilities in smart contracts include symbolic execution, fuzzing, formal verification, and pattern matching. These methods have their advantages but face several challenges, such as high resource consumption, limitations in detecting new types of

vulnerabilities, and difficulties in scaling to large contracts. As a result, there is a need to introduce new approaches, such as natural language processing (NLP) and machine learning, which can address these challenges more effectively. In this study, an NLP-based method was explored, using Word2Vec to convert smart contract code into vector representations, allowing for better analysis of the semantic relationships between elements of the code. These vector representations are then fed into a bidirectional recurrent neural network with GRU blocks and an attention mechanism. This approach allows the model to focus on the most important parts of the code and improve the accuracy of vulnerability detection. The comparative analysis showed that NLP-based methods significantly outperform traditional approaches in all key metrics. In particular, the GRU model with an attention mechanism demonstrated high results in accuracy, recall, and F-measure, making it effective for detecting complex vulnerabilities such as reentrancy. Furthermore, the NLP-based approach is capable of adapting to new types of attacks thanks to training on large datasets. Thus, the integration of NLP and machine learning represents a promising direction for enhancing the security of smart contracts. Future research can focus on improving these approaches, particularly through the implementation of advanced models such as transformers.

An integral part of any software project is the use of algorithms. Algorithms, which are the canvas, the basis of a large number of software solutions, are classical, printed in programmer's notebooks and often implemented in many programming languages. But any software project has many subtleties that cannot be exhausted by the classical approach, searches for ready-made solutions according to given parameters are also unsuccessful, or ready-made functions do not meet the given criteria of speed or accuracy. An example of such an algorithm, the solution of which is proposed in this work, is an algorithm for finding all minimum distances between any two vertices on a tree-structure graph. The Floyd-Warshall algorithm, universal, created for a graph of any structure, cyclic or directed, was used as the basis for its writing. The work proves that the tree structure of a graph is much simpler than a generalized cyclic structure, where the links can be edges or arcs. The work shows that in a tree there is only one path between any two vertices, that is, we do not have any need to recalculate the received answer - the distance between two specific vertices, hoping that it will be more optimal according to the criteria of the task, so we can get rid of a huge number of conditional operators and an outer loop, the number of iterations of which corresponds to the number of vertices, that is, if in a typical project the number of vertices of a tree-structure graph starts from a thousand vertices, then using the modified Floyd algorithm specifically for tree-structure graphs, and not the widespread, so-called, universal version, it is possible to speed up the work of the software product by at least a thousand times. In addition, tree-structure graphs have a number of structural differences that allow for another two to ten times of acceleration in relation to the ready-to-use universal version of the algorithm. So, the work considers a method for adapting the classical algorithm to specific needs, which are undirected tree-structure graphs.

Keywords: Algorithms on graphs; minimum distance search; optimization of calculations; influence of conditional operator on program performance; Floyd-Warshall algorithm; tree structure graphs